

The Transactor

🇨🇦 The Tech/News Journal For Commodore Computers

95% Advertising Free!

March 1986: Volume 6, Issue 05. \$3.50

Hardware and Software Interfacing

- Butterfield: Commodore 128 Machine Language Tips and Important Memory Map Locations •
- Writing Assembly Language Disk Access Routines • 16 Economical, Programmable, External Keys For Your C64 •
- Revealing The Programmable Logic Array • More On The Undocumented CPU Op-Codes •
- MIDI, For The Programmer • 3K RAM Expansion For The 1541 •
- Getting On Viewtron - It's Easy! •



THE TIME SAVER



J. MOSTACCI

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

Only \$7.95 Per Issue
6 Disk Subscription (one year)
Just \$45.00
(see order form at center fold)

Volume 6 Issue 05

Circulation 64,000

The Transactor

Hardware and Software Interfacing

Start Address Editorial 3

Bits and Pieces 5

C-64 Input Routine With Screen Editing!
Quick Screen Code to ASCII Conversion
C-64/VIC20 Mini-Datifier
Dale's Dazzler
Commodore 64 Meets The Alien From
The Cheap Sci-Fi Movie
VERIFIZER For Tape Users
Improved 1541 Head-Cleaning Program
PRINT AT Update
C-128 Bits
More B128 Bits From Liz Deal
Un-Scratcher For Commodore Drives
Hardware Device Number Change
C-64 Doodle Screen
1541 Write-Protect Check
C-64 Memory Fill ROM Routine
Relocate!

Letters 10

Oops - I Really Blew That One!
Pet Accessories: CP/M In The Transactor?:
A Coded Message: Medium Interest:
Monitored: SX Effects:
Wherever: Bloops Blues:
More Ad-vice: SuperPet Switch Glitch:

News BRK 76

Transactor Subscription Prices...
Viewtron Starter Kit
Viewtron Now Available To Commodore Owners
More Viewtron News
West Coast Commodore Show II
Commodore 128 On Dealer Shelves
The Commodore Ham's Companion
Starting Your Computer Services Business
1986 Printer Directory and Specification Guide
Scenery Disks Now Available for
Flight Simulator II and Jet
ZIPP-CODE-48 Development System For C64
Automated Telecommunications Package For
The 64 and 128
More 64 Software From Progressive
Progressive Distributes CBM 8023P Printers
Progressive Releases E-Link
Low Cost Temperature Monitoring For The 64
Communications Chips Seen Rising in Sales
As Semiconductor Industry Continues to Slow

The Verifier	Eliminate program entry errors	4
TransBASIC Installment #7		17
C128: Impressions and Observations		25
Maxims for the C128	Machine language tips from Jim Butterfield	26
Commodore 128 Memory Maps	Important locations	27
MIDI: Musical Instrument Digital Interface		28
The REL64 Cartridge	Interfacing to the real world	34
1541 RAM Expander	Add 3K of internal elbow room	36
Assembly Language Disk Access	I/O need not be slow	44
Directory Match	Take pattern matching one step further	48
C64 Memory Configurations	Secrets of the PLA	51
Undocumented 6510 Opcodes	One last look at the mystery ops	58
Undocumented 6500 Series Instructions	Er, two last looks	59
A Computer Rolltop Stand	Hiding clutter with class	61
Superkey	Build your own programmable number pad	64
String Calc	A number crunching demo in BASIC	66
Getting On Viewtron: It's This Easy!		68
The SAVE@ Saga	A brief update, more next issue	69
Compu-toons		69
The Transactor Disk Directory	Contents of Disks 1 through 9	71

Editor in Chief

Karl J. H. Hildon

Editor

Richard Evers

Technical Editor

Chris Zamara

Art Director

John Mostacci

Administration & Subscriptions

Lana Humphries

Contributing Writers

Ian Adam
Daniel Bingamon
Anthony Bryant
Jim Butterfield
Gary Cobb
Pierre Corriveau
Bob Davis
Elizabeth Deal
Michael J. Erskine
Jim Grubbs
Tom Hall
Bob Hayes
John Jay Hilfiger
John Holtum
Mark Jordan
Jesse Knight
James E. LaPorte
William Levak
Jack Lothian
Scott Maclean
Jim McLaughlin
Michael Mossman
Gerald Neufeld
Noel Nyman
Richard Perrit
Raymond Quirling
Glen Reesor
John W. Ross
Edward Smeda
Darren J. Spruyt
Nick Sullivan
Zoltan Szepesi
Karel Vander Lugt
Audrys Vilkas
Andrew Walduck
Jack Weaver
Charles Whittern

Production

Attic Typesetting Ltd.

Printing

Printed in Canada by
MacLean Hunter Printing

Program Listings In The Transactor

All programs listed in The Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter 'o' will of course be in lower case. Secondly, the lower case L ('l') has a flat top as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

print " flush right " - would be shown as - print "[10 spaces]flush right "

Cursor Characters For PET / CBM / VIC / 64

Down - q	Insert - I
Up - Q	Delete - i
Right - I	Clear Scrn - S
Left - [Lft]	Home - s
RVS - r	STOP - c
RVS Off - R	

Colour Characters For VIC / 64

Black - P	Orange - A
White - e	Brown - U
Red - L	Lt. Red - V
Cyan - [Cyn]	Grey 1 - W
Purple - [Pur]	Grey 2 - X
Green - I	Lt. Green - Y
Blue - -	Lt. Blue - Z
Yellow - [Yel]	Grey 3 - [Gr3]

Function Keys For VIC / 64

F1 - E	F5 - G
F2 - I	F6 - K
F3 - F	F7 - H
F4 - J	F8 - L

**Please Note: The Transactor has
a new phone number: (416) 878 8438**

The Transactor is published bi-monthly by Transactor Publishing Inc., 500 Steeles Avenue, Milton, Ontario, L9T 3P7. Canadian Second Class mail registration number 6342. USPS 725-050, Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to The Transactor, 277 Linwood Avenue, Buffalo, NY, 14209 ISSN# 0827-2530.

The Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64) are registered trademarks of Commodore Inc.

Subscriptions:

Canada \$15 Cdn. U.S.A. \$15 US. All other \$21 US.
Air Mail (Overseas only) \$40 US. (\$4.15 postage/issue)

Send all subscriptions to: The Transactor, Subscriptions Department, 500 Steeles Avenue, Milton, Ontario, Canada, L9T 3P7, 416 878 8438. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to Milton HQ. For best results, use postage paid card at center of magazine.

Editorial contributions are always welcome. Writers are encouraged to prepare material according to themes as shown in Editorial Schedule (see list near the end of this issue). Remuneration is \$40 per printed page. Preferred media is 1541, 2031, 4040, 8050, or 8250 diskettes with WordPro, WordCraft, Superscript, or SEQ text files. Program listings over 20 lines should be provided on disk or tape. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos or illustrations will be included with articles depending on quality. Authors submitting diskettes will receive the Transactor Disk for the issue containing their contribution.

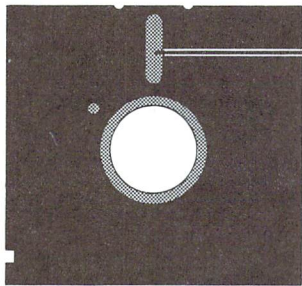
Quantity Orders:

U.S.A. Distributor: Capital Distributing Charlton Building Derby, CT 06418 (203) 735 3381 (or your local wholesaler)	Master Media 261 Wycroft Road Oakville, Ontario L6J 5B4 (416) 842 1555 (or your local wholesaler)	CompuLit PO Box 352 Port Coquitlam, BC V5C 4K6 604 941 7911	Norland Communications 631 Cloverpark Cres. Milton, Ontario L9T 4T7 416 878 8435
--	--	---	--

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 04, 05, 06, Vol 5 Issues 03, 04
Still Available: Vol. 4: 01, 02, 03. Vol. 5: 01, 02, 04, 05, 06. Vol. 6: 01, 02, 03, 04, 05

Back Issues: \$4.50 each. Order all back issues from Milton HQ.

All material accepted becomes the property of The Transactor. All material is copyright by Transactor Publications Inc. Reproduction in any form without permission is in violation of applicable laws. Please re-confirm any permissions granted prior to this notice. Solicited material is accepted on an all rights basis only. Write to the Milton address for a writers package. The opinions expressed in contributed articles are not necessarily those of The Transactor. Although accuracy is a major objective, The Transactor cannot assume liability for errors in articles or programs. Programs listed in The Transactor are public domain; free to copy, not to sell.



Start Address

Of Mice And Metaphors

We may be at a historical moment in the use of personal computers. Operating systems using mice, icons, desk-top metaphors, and windows seem to be catching on and only time will tell if they become the usual way of doing things in the future. We've seen it on Apple's Lisa and Macintosh, then the Atari 520ST, and now on the outstanding Amiga from Commodore. But is this a go-with-the-latest-computing-trend marketing effort, or a real step towards that difficult-to-measure goal of increased productivity?

Take the Icon. Please. Seriously, the concept of having a little picture on the screen instead of a word to represent a function is a good one, but can also be confusing. It is said that a picture is worth a thousand words, but that can be a problem when you want just one specific one. Does a jagged line icon in a graphics program indicate line drawing, graph plotting, straightening curves, or what? I suppose you could point to it to find out, but it would be nice to glance at the screen and know your options; key words like 'GRAPH', 'LINES', 'STRAIGHTEN', etc. would do the job nicely. Likewise, the word CUT may be clearer than a hard-to-visualize (especially on a low-res screen) pair of scissors. Words are more universal than pictures in the sense that not everyone interprets images in the same way; if you want proof, try playing "what do you see in the clouds?" with ten people. Today it may sound like a step backwards, but I often prefer good old-fashioned words over undecipherable symbols. Didn't mankind discover that thousands of years ago?

Throughout Canada, many of our road signs have gone symbolic. When you enter crown land, you may be greeted by a sign covered in little stick figures in various positions indicating the intended purpose of the land. If instead, the sign read, "hunting, fishing, camping, sightseeing. . ." perhaps it wouldn't be necessary to pull over, get out of the car and muse, "Hmmm. . . what's that guy doing? Is he using a shotgun or a walking stick? . . .". Similarly, the words SLOW and FAST on a piece of machinery are no doubt more easily understood than a picture of a tortoise and a hare, even if it means learning two words of English.

Which brings up one of the advantages of icons when used properly: they are international. Stress *when used properly*, because the temptation for software developers to transmit messages in a cute way through the icon is sometimes too great to resist. For example, The Amiga comes with a 'Software Demo Package' giving a sneak preview at some excellent-looking upcoming software from Electronic Arts. As the demo cycles, you can stop it at any time by pressing the right mouse button, and while the demo is stopped, an icon containing two - what are they? - um, pawprints appears on the screen. Get it? Paws - Pause! Ha ha. Cute and clever, and everyone gets a laugh when they figure it out or someone explains it to them, but that's the problem: are we now doomed to deciphering some unknown programmer's bad puns when all we want to do is use a piece of software? At the risk of

sounding like a spoilsport, there's nothing wrong with a bit of fun, but not at the expense of clarity and ease-of-use.

Enough about Icons; what about software metaphors themselves? The simulated desktop using 'windows' makes it easier for the non-computer type to get things done with a computer, because it relates to something he's familiar with. In a way, that's a big step forward because we are teaching machines to adapt to man rather than man to adapt to the machines. As a result, computer operation is becoming more intuitive rather than more complex. On the other hand, forcing a new office tool to behave like an old one in the interest of compatibility can be restrictive if the new tool is significantly more powerful and flexible. The ideal metaphor would probably mimic the human mind, not a human working environment, but that kind of software is no doubt many years off. Overall, the same rule applies to software metaphors as to icons: they can be good when used reasonably, but taken too far or used out of context, can be a hindrance.

I do like the windows themselves. At least, I like the way windows are handled on the Amiga - I haven't used any other system long enough to decide. True, you can lose track of what's behind what, and a lot of moving and clicking gets done, but it seems to be the best solution to keeping track of multiple tasks around. Still in its infancy, the windowed environment will probably get even more powerful and automated, and I'll like it even more.

What about the mouse? In my opinion, fun to use and precise, but isn't there a better way to point at something on the screen without clearing a space on your desk and *still* knocking over a cup of coffee once in a while? These things need S P A C E to be used effectively, and that's something that there can be a real shortage of - at least in my office there is. There's always the trackball, but it's not as easy to CLICK, since your hand doesn't stay in one place. The alternative, of course, is the keyboard, but I won't say I prefer it to the mouse because I'm still having too much fun with the little plastic rodent.

So, at the start of this new age of computing, does it look like utopia ahead? Will any non-computerist businessman or housewife be able to buy a computer and a bunch of software, and do all of the things that the ads have been telling them they could do? It seems that the new operating systems can deliver on that promise, but only if software developers use them in the spirit in which they were intended. When designed properly, a Mouse/Icon-driven Windowed environment can be easy to use and enormously productive. When used as a gimmick, or when flash prevails over substance, it can be an enigma to the user.

Chris Zamara, Technical Editor

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two-letter code for each program line which you can check against the corresponding code in the program listing.

There are two versions of VERIFIZER on this page; one is for the PET, the other for the VIC or 64. Enter the applicable program and RUN it. If you get the message, "***** data error *****", re-check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

SYS 828 to enable the C64/VIC version (turn it off with SYS 831)
or SYS 634 to enable the PET version (turn it off with SYS 637)

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing it means we've edited that line at the last minute which changes the report code. However, this will only happen occasionally and only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re-check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors (eg. POKE 52381,0 instead of POKE 53281,0), but ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm-start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

Listing 1a: VERIFIZER for C64 and VIC-20

```
KE 10 rem* data loader for "verifier" *
JF 15 rem vic/64 version
LI 20 cs=0
BE 30 for i=828 to 958:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50:
FH 60 if cs<>14755 then print "***** data error *****":end
KP 70 rem sys 828
AF 80 end
IN 100:
EC 1000 data 76, 74, 3, 165, 251, 141, 2, 3, 165
EP 1010 data 252, 141, 3, 3, 96, 173, 3, 3, 201
OC 1020 data 3, 240, 17, 133, 252, 173, 2, 3, 133
MN 1030 data 251, 169, 99, 141, 2, 3, 169, 3, 141
MG 1040 data 3, 3, 96, 173, 254, 1, 133, 89, 162
DM 1050 data 0, 160, 0, 189, 0, 2, 240, 22, 201
CA 1060 data 32, 240, 15, 133, 91, 200, 152, 41, 3
NG 1070 data 133, 90, 32, 183, 3, 198, 90, 16, 249
OK 1080 data 232, 208, 229, 56, 32, 240, 255, 169, 19
AN 1090 data 32, 210, 255, 169, 18, 32, 210, 255, 165
GH 1100 data 89, 41, 15, 24, 105, 97, 32, 210, 255
JC 1110 data 165, 89, 74, 74, 74, 74, 24, 105, 97
EP 1120 data 32, 210, 255, 169, 146, 32, 210, 255, 24
MH 1130 data 32, 240, 255, 108, 251, 0, 165, 91, 24
BH 1140 data 101, 89, 133, 89, 96
```

Listing 1b: PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI 10 rem* data loader for "verifier 4.0" *
CF 15 rem pet version
LI 20 cs=0
HC 30 for i=634 to 754:read a:poke i,a
DH 40 cs=cs+a:next i
GK 50:
OG 60 if cs<>15580 then print "***** data error *****":end
JO 70 rem sys 634
AF 80 end
IN 100:
ON 1000 data 76, 138, 2, 120, 173, 163, 2, 133, 144
IB 1010 data 173, 164, 2, 133, 145, 88, 96, 120, 165
CK 1020 data 145, 201, 2, 240, 16, 141, 164, 2, 165
EB 1030 data 144, 141, 163, 2, 169, 165, 133, 144, 169
HE 1040 data 2, 133, 145, 88, 96, 85, 228, 165, 217
OI 1050 data 201, 13, 208, 62, 165, 167, 208, 58, 173
JB 1060 data 254, 1, 133, 251, 162, 0, 134, 253, 189
PA 1070 data 0, 2, 168, 201, 32, 240, 15, 230, 253
HE 1080 data 165, 253, 41, 3, 133, 254, 32, 236, 2
EL 1090 data 198, 254, 16, 249, 232, 152, 208, 229, 165
LA 1100 data 251, 41, 15, 24, 105, 193, 141, 0, 128
KI 1110 data 165, 251, 74, 74, 74, 74, 24, 105, 193
EB 1120 data 141, 1, 128, 108, 163, 2, 152, 24, 101
DM 1130 data 251, 133, 251, 96
```


Bits and Pieces

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits & Pieces column, we'll credit you in the column and send you a free one-year's subscription to The Transactor

C-64 Input Routine With Screen Editing!

**Dale Lambert,
Tupelo, MS**

This little INPUT substitute allows any characters to be entered and also allows full screen editing. It uses the input routine that BASIC uses in direct mode.

```
1 sys42336:forb=512to592:ifpeek(b)<>0thennext
2 in$=" ":poke peek(71)+256*peek(72)+1,0
  :poke peek(71)+256*peek(72)+2,2
3 pokepeek(71)+256*peek(72),b-512
  :in$=mid$(in$,1)
```

Quick Screen Code to ASCII Conversion

Dale Lambert

This line will convert screen code (in the variable S) to ASCII:

```
a=(s and 127)or((s and 64)*2)or((64-s and 32)*2)
```

C-64/VIC20 Mini-Datifier

Dale Lambert

This program will quickly and easily make DATA statements for a machine-language program.

All you have to do is put the starting address of the code in variable S, put the end address in E, put the starting line number of the DATA statements in Z, and the amount to increment the DATA line numbers by in variable I. For example:

```
1 s=49152(start):e=50000(end):z=1000(line #):i=10(incr)
```

And here's our program:

```
1 s=49152:e=49400:z=1000:i=10
2 print "Sqq" "z" data " ":if s>e then end
3 k=s+6:if k>e then k=e
4 for s=s to k:print mid$(str$(peek(s)),2) ", " :next
  :print chr$(157):chr$(32):rem 1 left, 1 spc
5 print "s="s":e="e":z="z+i":i="i":goto2s";
  :poke631,13:poke632,13:poke198,2:end
```

Dale's Dazzler

Try this on the 64:

```
1 a=192:b=200:c=53270:fori=1to1000step.001
  :pokec,a:pokec,b:next
```

Commodore 64 Meets The Alien From The Cheap Sci-Fi Movie

Giuseppe Amato

Give this a listen, Earthlings:

```
1 s=54272:a=peek(162)and199:pokes+24,15
  :pokes+6,90:pokes+4,21:pokes+1,a
2 pokes+15,abs(99-a):goto1
```

VERIFIZER For Tape Users

**Tom Potts
Rowley, MA**

The following modifications to the Verifier loader (see the VERIFIZER page in this issue) will allow VIC-20 and C-64 owners with Datasets to use the verifier directly (without the loader) and just SYS to activate it.

After running the new loader, you'll have a special copy of the verifier program which can be loaded from tape without disrupting the program in memory.

Just run the program below, pressing PLAY and RECORD when prompted to do so (use a rewind tape for easy future access). To use the special verifier that has just been created, first load the program you wish to verify or review into your computer from either tape or disk. Next insert the special program tape created above and be sure that it is rewind, then enter in direct mode: OPEN1:CLOSE1. Press PLAY when prompted by the computer, and wait while the special verifier loads into the tape buffer. Once it has loaded, the screen will show FOUND VERIFIZER.SYS850. To activate VERIFIZER, enter SYS 850 (not the 828 as in the original program). To de-activate, use SYS 853. These moves in the SYS addresses were required because of the method used to store and retrieve the program in the tape buffer.

If you are going to use your tape recorder to SAVE a program, you must turn off VERIFIZER first (SYS 853) since VERIFIZER moves some of the internal pointers used during a SAVE operation. Attempting a SAVE without turning off VERIFIZER first will usually result in a crash.

If you wish to use VERIFIZER again after using the tape, you'll have to reload it with the OPEN1:CLOSE1 commands.

Make the following additions and changes to the present VERIFIZER loader listed in the magazine on page 4.

```

Line:
NB 30  for i=850 to 980: read a: poke i,a
AL 60  if cs<>14821 then print "*****data error*****"
      : end
IB 70  rem sys850 on, sys853 off
-- 80  delete line
-- 100 delete line
OC 1000 data 76, 96, 3, 165, 251, 141, 2, 3, 165
MO 1030 data 251, 169, 121, 141, 2, 3, 169, 3, 141
EG 1070 data 133, 90, 32, 205, 3, 198, 90, 16, 247
BD 2000 a$ = "verifier.sys850[space]"
KH 2010 for i=850 to 980
GL 2020 a$ = a$ + chr$(peek(i)):next
DC 2030 open 1,1,1,a$: close 1
IP 2040 end

```

Improved 1541 Head-Cleaning Program

David Peterson,
Irvine, CA

Volume 6 Issue contained a program by Peter Boisvert which turned the 1541's motor on for 60 seconds to allow cleaning the head using a cleaning disk. This prompted David Peterson to write in with the following improvement. It turns the motor on, then steps the head slowly along the surface of the disk to utilize the entire cleaning surface. David Peterson explains how it works:

After turning on the drive motor, the program peeks location \$24 in drive RAM. This location contains the track number that the read/write head is currently at. After finding the head, the program steps it quickly to track 1, then slowly across the disk to track 35. Movement of the head is controlled by bits 0 and 1 of location \$1C00 in drive RAM. After peeking \$1C00, the head is moved outward to track one by cycling bits 0 and 1 of \$1C00. To move the head outward the low bits are decremented (say 01 to 00 to 11 to 10 to 01 etc.). To move the head inward to track 35 the two low bits are cyclically incremented. The head is stepped twice for every track, since the stepper motor moves the head in 1/2 track steps. The NEW at the end of the program is not an attempt at program protection, it's there as drive protection. This direct method of stepping the head does not update location \$24. If the program was immediately rerun, the drive head could end up being stepped to track 35 or to bump up against the stop at track 0. Therefore use the loop in line 280 to control how long the process takes.

Here's the new cleaning program; make sure you save it before running!

```

LM 100 rem* improved 1541 head cleaning prg *
LD 110 print "SInsert cleaning disk and hit return "
NI 120 geta$:ifa$<>chr$(13)then120
AL 130 open 15,8,15:print#15,"m-e" chr$(126)chr$(249)
EE 140 rem locate head
HP 150 print#15,"m-r" chr$(24)chr$(0)
JD 160 get#15,a$:x = asc(a$ + chr$(0))
EC 170 print "drive head at track #" x
IA 180 rem read $1c00
PH 190 print#15,"m-r" chr$(0)chr$(28)
EL 200 get#15,sc$:sc = asc(sc$ + chr$(0))
FO 210 rem select bits 0 and 1
GN 220 bt = sc and 3
CD 230 rem # tracks to 1
ID 240 sp = 2*(x-1)
MH 250 rem move head to track 1
BL 260 print "qstepping to track #1 "
FJ 270 for y = 1 to sp
EF 280 bt = bt-1:bt = bt and 3
KO 290 s = (sc and 252)or bt
IH 300 print#15,"m-w" chr$(0)chr$(28)chr$(1)chr$(s)
FE 310 next y
HL 320 rem step out to 35
IM 330 print "stepping out to track # 35. . ."
FB 340 print#15,"m-r" chr$(0)chr$(28)
CH 350 get#15,a$:sc = asc(a$ + chr$(0))
CG 360 bt = sc and 3
CE 370 for y = 1 to 68
GF 380 print "qq track # " int(y/2 + 1)
AF 390 print "QQQQ"
JM 400 bt = bt + 1: bt = bt and 3
CG 410 s = (sc and 252) or bt
AP 420 print#15,"m-w" chr$(0)chr$(28)chr$(1)chr$(s)
FF 430 ford = 1to220:nextd
HM 440 next y
FI 450 print#15,"m-e" chr$(232)chr$(249):close15
HF 460 new: rem to prevent re-running without a normal
      disk operation first

```

PRINT AT Update

Stephen Gast, Champaigne. IL

In the Bits and Pieces column of Volume 6, Issue 3, a C64/VIC20 PRINT AT command was suggested:

```
poke 781,row: poke 782,col: sys 65520: print "message"
```

The above method utilizes the documented KERNAL routine PLOT. The general technique is a useful one but can be unreliable when accessed through the KERNAL jump table at 65520 (\$FFF0).

If the carry flag is set, the routine will GET the current cursor position! Not exactly what we had in mind. To correct this the current row and column coordinates should still be placed directly into the register storage area in 781 (\$030D9 and 782 (\$030E). Then simply bypass the logic of the cursor get/set routine at \$E50A and SYS directly to 58636 (\$E50C). In both the VIC 20 and the Commodore 64 this will work:

```
poke 781,row: poke 782,col: sys58636: print "message"
```


Now let's talk a little about some other things you can do on a 64. First, the following line is an alternative to the above example:

```
poke 211,col: poke 214,row: sys 58640: print "message"
```

This enters the plot routine a little later and avoids two steps (big deal at ML speeds). But secondly, if you do this a lot in a program, here is a neat 25 byte machine language routine that makes life a little simpler:

```
0806 20 fd ae jsr $ae fd ;scan past the comma
0809 20 9e b7 jsr $b7 9e ;put row in .X
080c 86 d6 stx $d6 ;store row in TBLX (current cursor
line #)
080e 20 f1 b7 jsr $b7 f1 ;scan past comma and put column
in .X
0811 86 d3 stx $d3 ;put column in PNTR (current cur-
sor column #)
0813 4c 10 e5 jmp $e5 10 ;set the cursor
```

The following short BASIC program will place this routine in a REM statement:

```
10 rem xxxxxxxxxxxxxxxxxxxxxxxxxx [25 x's]
20 for x=2054 to 2069: read y: poke x,y: next x
30 data 32,253,174,32,158,183,134,214,32,241,183,
134,211,76,16,229
```

Be sure 25 x's follow the REM in line 10. After typing the program in, run it and delete lines 20 and 30. Save the remaining line 10 to disk as a program and simply load and use it as the first line of any program in which you want to easily be able to position the cursor. The syntax is now simplified to:

```
SYS 2054,row#,column#: PRINT "message"
```

C-128 Bits

Randy Linden, Willowdale, Ont.

Here are a couple of "C64 mode" peculiarities: The CAPS-lock key can be read in C64 mode, and the most interesting feature - 2Mhz. clock speed is available in C64 mode!

The CAPS-lock key can be checked with bit 6 in memory location \$0001 of the C64 side. If bit 6 is set, then the CAPS-lock key is NOT pressed; if it is zero, the key is pressed. Example:

```
if (peek(1)and64)=0 then print "caps lock on".
```

Bit 0 in memory location \$D030 controls the speed of the micro-processor. In C64 mode, this bit is normally zero, running the system at 1Mhz. If you set this bit, the computer will run at 2Mhz! Example:

```
poke 53296,peek(53296)or1
```

to set 2Mhz mode. The catch is that in C64 mode, the VIC video chip cannot operate at warp 2 and is disabled when 2Mhz mode is set, displaying a blank screen. However, for operations which do not need the video screen, such as assembling programs in

machine language or sorting lists, the screen can be turned off for a speed increase of 100%!

For a simple demonstration, try the following program in C64 mode.

```
10 print "C64 at 2Mhz - Randy Linden"
20 print "Caps-lock down for 2Mhz,"
30 print "Caps-lock up for 1Mhz."
40 for d=0 to 36
50 read rl: poke 52992+d,rl: ck=ck+rl: next d
60 if ck<>3571 then print "?Data error!": stop
70 sys 52992: print "Now installed.": end
100 data 169, 11, 141, 20, 3, 169, 207, 141,
21, 3, 96, 165, 1, 41, 64, 73, 64, 10, 10, 42
110 data 141, 37, 207, 173, 48, 208, 41, 252,
13, 37, 207, 141, 48, 208, 76, 49, 234
```

The code resides at \$CF00-\$CF25 in memory on the Commodore 64. When run, it changes the IRQ vector to point to a routine at \$CF0B which scans the CAPS-lock key and turns on 1Mhz mode if it is up, or 2Mhz mode if it is down.

More B128 Bits From Liz Deal

1. COLLECT is a variant of the DOS native command VALIDATE (V). The B machine actually *thinks* at the time of validation. Example: if you've just written a file of 6 blocks but didn't close it, the directory shows 6 blocks and a *. At this point "V" would get rid of the file, but COLLECT closes it. Not bad.

2. An important control byte exists at \$258: Logical file number of CMD file. It's used for printing integers (line numbers) on a CMD device. It permits sending disk directories directly to a CMD device, so:

```
OPEN 4,4:CMD 4: CATALOG
```

Does just that to the printer. Neat.

3. everybody knows that LIST is a harmless command. . .or is it? Try it on the B128 with a program from the plus 4 containing the keyword SCALE (that's token number \$E9). TThe machine either crashes dead or ends up in the machine language monitor. And if that weren't enough, the program you just tried to list NEWs itself. Brilliant.

4. Locations \$20-\$21 are important in working dynamic strings. They hold temporary pointers to strings. Do not ever use them for anything else if you do 't want your strings mangled, FRE crashing, and so on.

5. New center of B information, besides TPUG, is now:

Norman Deltzke
4102 N. Odell
Norridge, IL
60634
- Send 2 SASEs.

Un-Scratcher For Commodore Drives

Oops! Just scratch the wrong file by mistake and wipe out 3 hours of work? Don't panic – use the Un-Scratcher. If you haven't done any more saving since the scratch, your old file is still recoverable. The Un-Scratcher program below will ask you the type of drive you have (you may want to hard-code this into the program if you're always using the same drive), and display the filenames of scratched programs one by one, asking if you wish to un-scratch them. It will then write in the new directory information and validate the disk, asking for confirmation before each step. After that your life may continue normally since your precious work has been restored. Even if you have to enter it by hand, this is a routine that can really pay for itself!

```

CN 100 rem save "0:un-scratch ",8: rem ** rte/85
NL 110 z$ = chr$(0): cr$ = chr$(13): sc = 1: dr = 0
    : rem dirsec + drvnum
FI 120 print "** disk file un-scratcher **" cr$ "enter drive type"
LO 130 print " a) 1541/2031 " cr$ " b) 2040/4040 " cr$ " c) 8050/8250
    "
FM 140 input "your choice ";dt$: if dt$ < "a" or dt$ > "c" then 140
MP 150 dt = 18: bh = 3: if dt$ = ">" b" then bh = 17
    : if dt$ = "c" then dt = 39
KE 160 open 15,8,15: open 8,8,8, "#0"
EN 170 print#15, "u1: " 8;dr;dt;sc: rem read dir sector
GD 180 print#15, "m-r" chr$(0)chr$(bh)chr$(2): get#15,nt$,ns$
    : rem next trk/sec
MG 190 flag = 1: for lk = 0 to 7: ps = lk*32
    : print#15, "m-r" chr$(2 + ps)chr$(bh)chr$(19)
KD 200 get#15,sb$,ft$,fs$: if len(sb$) or len(ft$) = 0 then 310
LE 210 print "1st data track" asc(ft$) "sector" asc(fs$ + z$)
BD 220 print "filename " ;: for name = 1 to 16: get#15,n$: print n$;
    : next
FF 230 print#15, "m-r" chr$(30 + ps)chr$(bh)chr$(2): get#15,l$,h$
    : rem file size
LE 240 print: print "size" asc(l$ + z$) + 256*asc(h$ + z$) "block(s)"
KD 250 input "un-scratch (y/n) ";us$: if us$ <> "y" then 310
    : rem * nope
KN 260 input "file type: s, p, u, r ";ft$: us = 0
ML 270 for chk = 1 to 4: if ft$ = mid$("spur",chk,1) then us = chk + 128
OK 280 next chk: if us = 0 then 260: rem incorrect reply
KM 290 print#15, "m-w" chr$(2 + ps)chr$(bh)chr$(1)chr$(us)
CH 300 print "done !!": print: flag = 0
KE 310 next lk
EB 320 if flag then 350: rem no change in sector
AH 330 input "write block to disk (y/n) ";yn$: if yn$ <> "y" then 350
MK 340 print#15, "u2: " 8;dr;dt;sc
PA 350 dt = asc(nt$ + z$): sc = asc(ns$ + z$): if dt then 170
    : rem more to go
NE 360 if us = 0 then 390: rem nothing to un-scratch
OG 370 input "validate the diskette (y/n) ";yn$: if yn$ <> "y" then 390
CO 380 print#15, "v" + str$(dr): print ">> validating disk <<"
PF 390 close8: close15: end

```

Hardware Device Number Change for the 2031 Drive

The 2031 single drive can be hard-wired as device number 9 or 10. The number is determined by two diodes on the PC board, CR18 and CR19. Both diodes are normally connected for device number 8; snip one of the leads on CR18 for device 10 or CR19 for device 9.

C64 Doodle Screen

**Tom White,
Sudbury, ON**

If you've ever drawn pictures on the screen with the standard graphics and editor on the C-64, you've probably hit the RETURN key by accident more than once. This results in a READY or ?SYNTAX ERROR message partly wiping out your creation. To counter this menace, simply enter the following before you start your masterpiece:

```
poke 768,123: poke 769,164
```

While in this mode, all of the BASIC commands still work, so take care not to type LIST, RUN or any similar instruction that might ruin your picture. To return the error messages back to normal, type:

```
poke 768,139: poke 769,227
```

1541 Write-Protect Check **Craig McQueen, Guelph, ON**

Have you ever wanted a routine to find out if there is a write-protect switch on a disk? All one has to do is read the value of \$1C00, bit 4. If the bit is 0, then the write protect is on. Here is a memory-read routine to do the checking for you.

```

5 rem check for write-protect (1541)
10 open 15,8,15
20 print#15, "m-r" chr$(0);chr$(28)
30 get#15,a$: a = asc(a$ + chr$(0))
40 if(a and 16)then 70
70 print "write protect is on!"
60 goto 80
50 print "no write protect"
80 close 15

```


C-64 Memory Fill ROM Routine

Thomas Henry
North Mankato, MN

Thomas Henry writes:

In the Volume 6, Issue 1 Bits & Pieces section, you described the use of the memory transfer subroutine contained within the BASIC ROM. One vital piece of information is missing, though. The memory transfer routine is not "intelligent". Specifically, it fails to work correctly if you attempt to move a block of data in the downward direction AND if the source block overlaps the destination block. In all other cases however, it works just fine.

By the way, the routine has this limitation since it was originally designed to spread apart BASIC lines in memory. The designers apparently never intended for it to be used for any other purpose than making room for new BASIC lines in RAM. Of course, this type of memory transfer will always be occurring in an upward direction (from lower to higher addresses). Now that we know the limitation, however, we hackers can use it for other purposes as well.

Here's a neat trick that actually exploits the shortcomings just mentioned to good advantage. See if you can figure out how it works. (The following addresses are for the C-64; refer to the abovementioned issue for the corresponding addresses in the PET/CBM and VIC-20.)

THE TASK: We wish to fill a block of RAM with a specific byte. We'll call this a memory fill subroutine. Possible uses include clearing a bit-map screen, setting colour memory to some value, filling a buffer with zeroes, etc.

THE SOLUTION: Suppose the addresses of the start of the block to be filled is START and the last address of the block is END. For example sake, let's imagine we wish to fill this block with zeros. Perform the following steps:

1. Store a zero (or whatever byte you wish to fill memory with) in location END
2. Store address END in location \$58 (low byte, high byte).
3. Store address END + 1 in location \$5A.
4. Store address START + 1 in location \$5F
5. Call subroutine \$A3BF (The memory transfer subroutine)

The block from START to END (regardless of length or location) will be filled with the specified byte - zeros in this case.

One limitation of the block fill routine is that it cannot be accessed from BASIC. Apparently the POKE number or SYS number evaluator plays havoc with locations \$58 through \$60. However, the routine works just great with machine language and is far simpler to use than "rolling your own".

Relocate!

When creating sprite files, high-res screens, character sets and the like, you don't always know where in memory you'll want to put them. You'd also like those files to be LOADable programs that will go into whatever spot you want. For example, some drawing packages create a high-res screen PRG file on disk that can be LOADED into memory starting at \$2000 (8192). If you wish to use that picture but put it at, let's say, \$E000, you need to change the picture file's load address on disk. Guess what RELOCATE does? Just tell it your drive type, the filename of the program, and your desired load address, and it changes the load address of the file. Right now. Yes, it is fast, because it goes directly into disk memory to change the first two bytes of the file and writes the block back to the disk surface.

```

HD 100 rem save "0:relocate",8
AF 105 rem ** rte/85 - allows quick change of prg load
    address **
CO 110 :
GK 115 z$ = chr$(0)
CK 120 print " drive type: "
GH 125 input " 1)1541/2031, 2)4040, 3)8050/8250 ";d
    : if d<1 or d>3 then 125
DD 130 if d = 1 then dl = 144: dh = 2: di = 4: dt = 18: bl = 0
    : bh = 3: rem 1541/2031
HC 135 if d = 2 then dl = 150: dh = 67: di = 4: dt = 18: bl = 0
    : bh = 17: rem 2040/4040
OF 140 if d = 3 then dl = 96: dh = 67: di = 8: dt = 39: bl = 0
    : bh = 17: rem 8050/8250
FA 145 :
AJ 150 input " drive #, filename ";dr,f$
    : f$ = str$(dr) + " " + f$
HE 155 input " new start address (decimal) ";sa
    : sh% = sa/256: sl = sa - 256*sh%
MP 160 open 15,8,15: open 8,8,8,(f$): get#8,a$
    : if st then close8: stop
FL 165 print#15, " m-r " chr$(dl)chr$(dh): get#15,s$
    : rem sector
IP 170 print#15, " m-r " chr$(dl+di)chr$(dh): get#15,i$
    : rem index
ID 175 s = asc(s$ + z$): i = asc(i$ + z$) + 1
II 180 close 8: open 8,8,8, "#0"
AH 185 print#15, " u1: " 8;dr;dt;s: rem read in directory
    track/sector
BD 190 print#15, " m-r " chr$(bl+i)chr$(bh)chr$(2)
    : rem get 1st data block ptr
OM 195 get#15,t$,s$: t = asc(t$ + z$): s = asc(s$ + z$)
HF 200 print#15, " u1: " 8;dr;t;s: rem get first data block
DM 205 print#15, " m-w " chr$(bl+2)chr$(bh)chr$(2)
    chr$(sl)chr$(sh%)
BP 210 print#15, " u2: " 8;dr;t;s: rem write block back
IE 215 close 8: close 15
GE 220 print " ** address changed ** "
BO 225 end

```


Letters

Oops – I Really Blew That One! *Within minutes after the last issue was released, or so it seemed, I ended up on the receiving end of a telephone conversation with Chris Wiesner in Ottawa. The conversation we had made me blush for the first time in years. It seems that in his letter, as we reproduced it in on page 13 of last issue, I really fouled up – twice. The first was to spell his name incorrectly (I before E except after C). The second, and this was pretty awful, was to incorrectly state the telephone number of his Bulletin Board System in Ottawa. Chris, in a blast of ingenuity, was going to contact the people belonging to the number listed, and lend them his answering machine. Can you imagine how rotten it would be to have your phone ring constantly through the night, only to hear a computer screaming at you when you pick it up. Pretty bad. Anyway, sorry Chris, and multiple sorry to the people who belonged to that line. Unlike my 'The Error Of Our Ways: More Often Oops Than Bloops' note from last issue (page 14), of which Chris made sure to mention, we do foul up at times. Below is the proper name and address of Chris's BBS. Ed.*

Chris K. Wiesner
Ottawa Mail Forwarding Service
PO Box 793 Station B
Ottawa, Ontario, K1P-5P8
(613) 830-2823

Pet Accessories: I am the proud owner of a Pet 4032 computer and know several others in my area. Our problem is that there is so little support for the Pet in this area. In fact, nobody has even heard of it! I have need of certain information that I hope you will help me get.

There is an upgrade kit available for the 4032 that is supposed to turn it into an 8032. This is from Comspec Communications Inc. in Toronto, Ontario. Does this really convert the 4032 to an 8032? Also, does this conversion effect the compatibility of the upgraded Pet to existing software? I have also heard of an upgrade kit which adds 64k of memory to the Pet. Where is this available? The Commodore SFD 1001, one megabyte disk drive has recently arrived on the scene unannounced, with little fanfare and even less technical data. It is available from Protecto and Progressive Peripherals. Is this drive software compatible with other Commodore drives? Which ones? What commercial software is available for the Pet in either the 40 or 80 column configurations? What drive is this software designed for?

William Uhler, Richmond, VA

For such a small letter, you sure squeeze in a lot of questions. First, Comspec does sell an update kit for the 4032 to turn it into a true 8032. This update also allows the user to switch back and forth between 4032 and 8032 mode. The trick with this update is that the 4032 has a different keyboard than that of the

8032. To get around this problem, they have written in a control key sequence that allows the user to produce any normal CBM key sequence desired. For an added bonus, quite a few useful but normally unavailable special effects can also be produced. This conversion will not affect the execution of any program written specifically for the 4032 or 8032. The machine will behave exactly as expected. The address for Comspec follows this reply.

In reply to the myth of an extra 64k board for the Pet, it is partially true. First, the Pet has to become an 8032. From that point on the extra 64k is no problem, providing you are willing to shell out \$399.00 Cdn. The 64k in question is really the extra board provided with a Commodore 8096 computer, originally designed in Great Britain. The 64k is mapped in a really strange way, though. It overlays screen memory and ROM when enabled, and cannot be used from within Basic. Although this may appear to be a major obstacle in your way, it may not be. A few software packages have been written for the 8096. For example: Wordpro 5+, Paperclip Expanded, Calc Result, and Silicon Office. For a laugh, VisiCalc is also available through the bootleg community for the 8096. Although VisiCorp originally wrote Pet and CBM versions years ago, they had a fallout with Commodore before cutting a version for the 8096. Enter the mysterious hacker. Not only was the package broken (originally ROM protected), but it was also converted over to the 8096. Speaking from personal experience, the package is Ok. Lots of user workspace, and no apparent bugs. I'm sure VisiCorp wasn't too happy with that one.

The SFD 1001, as mentioned last issue, is completely read/write compatible with the now defunct Commodore 8250 drive. With a little bit of effort, and keeping in mind the fact that the SFD 1001 uses both sides of the diskette, the Commodore 8050 is read/write compatible. The 8050 uses only one side of the diskette, 77 tracks – 540k, therefore, when an 8050 diskette is used in an SFD 1001 drive, it will have to be initialized once before access can begin. The first attempt at initialization will always err out, but any access thereafter will be Ok. If put in the reverse situation, SFD 1001 diskette in a 8050 drive, it is completely read/write compatible, providing that no programs to be accessed span onto the opposite side of the diskette. One last note to mention before continuing. The SFD 1001 is a parallel drive, ie. IEEE, therefore anyone wanting to use this beast with their 64 will also need to purchase an IEEE interface.

As a final wrap up of your letter, there is quite a bit of really fine, and not so fine, software available for both the Pet and CBM machines. If you were to prowling around a few of the older user group libraries, you should find plenty of Pet stuff. Contact TPUG and ask if they can help you with this one. As far as commercial software goes, there are quite a few packages available, but locating them may be a bit of a problem now.

Packages such as Superscript, Paperclip, Wordpro, Wordcraft, VisiCalc, Calc Result, The Manager, Jinsam, Ozz, Pal, Power, plus great gobs more, are around. Try finding a copy of the latest 'Commodore Software Encyclopedia', published by Sams Books, and take a peek inside. They not only have listings of recent software for the newer Commodore machines, but also have pages of listings of Pet/CBM software, complete with descriptions, drive and memory requirements, plus distributors names and addresses.

Comspec
153 Bridgeland Avenue, Unit #5
Toronto, Ontario, Canada
M6A-2Y6 (416) 787-0617
Attention: Nick Hooper

A Coded Message:

Dear Sirs

```
PROGRAM LETTER (INPUT, OUTPUT);
TYPE SCALE OF [1..10];
  DECUS : FILE OF INFORMATION FOR DIGITAL EQUIPMENT COMPANY
  TRANSACTOR : FILE OF INFORMATION;
CONST VALUE-OF-TRANSACTOR = 10 ON SCALE;
  ISSUE-NOVEMBER = 8 ON SCALE;
VAR COMOL, MACHINE : ABUSED;
  C ; ENJOYABLE;
  GAZETEER OF LANGUAGES : CHAR FROM DECUS;
  LOGO : PLEASE;
  DONE : BOOLEAN;
  DR. ROSS : THANK YOU;
BEGIN;
  DONE := FALSE;
  WHILE NOT DONE DO;
  BEGIN; (*DECUS*)
    READLN (DATE-PUBLICATION);
    WRITELN('Parts of',GAZETEER OF LANGUAGES,' were in',DATE-PUBLICATION);
  END; (*DECUS*)
RESET DECUS;
  BEGIN; (*OTHER STUFF*)
    WRITELN('Where is the Pascal, Ada, Fortran and Promal?');
    READLN(WHERE IS THE BEEF);
    WRITELN('Maybe programmers should learn an other language, ENGLISH');
  END; (*OTHER STUFF*)
LOGO := 'Why noy use Pilot?'
  BEGIN; (*NATIONAL COMMODORE LANGUAGE INTEREST GROUP*)
    IF READER = INTERESTED IN OTHER PROGRAMMING LANGUAGES
    THEN BEGIN;
      WRITELN('Contact NCLIG');
      WRITELN('1812 N. I. Street');
      WRITELN('Fremont, NE 68025');
      WRITELN('[(402) 721-4346]');
    END;
  END;
  DONE := TRUE;
END.
```

Kent Tegels, Fremont, Nebraska

We get some of the most unusual letters, but never before one as unusual as this. For everyone reading, this letter has been reproduced exactly as it was sent to us.

Sorry for not covering as many languages as hoped for in our Languages issue, but it wasn't due to a lack of effort. For quite some time prior to the Languages issue, we chased about trying to locate people who would be willing to write about the various languages available. For our efforts we received many promises, but few results. Such is life. The issue was still good, but did not cover the languages as extensively as hoped.

About Promal. I contacted their distributor a fair time before the Languages issue was to begin, but received the package after the issue rolled off the press. The answer to this puzzle came to me while using the package. The compiler's date stamp was just days prior to my receiving it. The package wasn't completed until after we finished the issue! Not to rain on Promal's day altogether, it seems to be a terrific package, one worthy of review in some future issue.

And finally, I like your tidy bit of advertising. If some of you missed it near the bottom of the letter, here it is again. If you are interested in other programming languages, you can contact the National Commodore Language Interest Group (NCLIG) at 1812 N. I. Street, in Fremont, NE, Zip 68025, tel# (402) 721-4346.

Monitormented: I am the proud owner and user of Commodore machines, but I'm having trouble with one of their animals, namely the 8032 Assembler/Monitor. You see, I don't have the users manual for the creature and regardless what commands I feed it – .A, .D., .R, etc. – it burps up the "?". Very aggravating! I've tried both decimal and hex for starting addresses, accumulator values, etc., to no avail. One of you guys must know how to feed it. Please HELP.

Randy Guimond, Baie Ste. Anne, New Brunswick

It sounds like you're referring to the Machine Language Monitor that resides in the 8032 ROMs. If so, it doesn't do a whole lot. It's useful, but limited. It has no Assembler, or Disassembler, or any of the more sophisticated commands found in many MLM extension programs available. Here is the command set of the resident MLM (remember, the period preceding the command is supplied by the machine).

- .G Go to a specific address of code, or whatever is in the PC
- .L Load a file – syntax L "drive:filename",device# (ie.08)
- .M Display computer memory in hex. Syntax; .M ssss eeee ; where ssss is the start 16 bit hex address to view, and eeee is the ending 16 bit address.
- .R Displays registers - alter by typing new hex values over the existing ones, and hit return
- .S Save a file – syntax S "drive:filename",device # (ie. 08),start,end + 1
- .X Exits to Basic

However, it's also possible you're referring to a program you are loading, perhaps from a diskette. As mentioned above, this would be a monitor extension program. There are many of these – Extramon, Supermon, Micromon, to name a few – that link themselves into the resident MLM for extra commands like Hunt, Fill, Walk, and many more. If this is the case then I'm not sure which extra commands you may or may not have. If there are any, a sure clue is the Disassemble command – I can't think of any extension that doesn't include it. (Try .D followed by any 16 bit hex address) Get a copy of Extramon or Micromon, all in public domain, and then try your luck. They are guaranteed Ok.

Whereware: To begin this letter I must congratulate you on what I consider the best magazine currently available which covers the Commodore line of computers! PLEASE keep up the good work and keep the magazines coming!

In the September issue (1985) of The Transactor, there was a letter from Real Gagnon, of Quebec, asking about a 40/80 column adapter for the VIC-20 from Data-20. As of February of this year, Data-20 (at the address you listed) still had their 40/80 column adapter with 16k memory, at a close-out price of \$70.95 including shipping. When I ordered one for my VIC, I got the impression that they were not moving very fast, so there may still be some left.

THANKS for the assembler program for the Rockwell R65C02 chip! I have also recently purchased and installed one in my VIC, and now I can write software for it. Now all I have to do is to re-write the screen editor in the Data-20 40/80 column adapter to utilize the new op-codes – and to correct some of the glitches in the editor.

Also, in regard to the letter from Bill Uhler, of T.R.A.C.E., I agree that when the distributor or publisher of software is no longer in business, it is a pain in the tutu. Keep in mind that often a copyright is only assigned to the marketing company for the period that it is in business, and usually reverts to the author upon the demise of the marketing company. Have you considered starting a clearinghouse, to keep track of the authors and publishers of software for Commodore computers? You could either get permission from the copyright holder to produce the software yourself, and charge for the copying, or you could maintain a database of copyright holders, and sell lists of the names. If enough people wrote to the author of a program to request permission to copy it, perhaps the author would be convinced to either start publishing it again, or release it to the public domain.

To add my own 2-cents worth to the copy protection controversy: I can well understand the viewpoint of the authors and 'publishers' of software, but I am also a buyer of software. There is only one pirate-prevention method which I have any respect for – the 'dongle' system as used by Batteries Included on their Consultant and Paperclip software packages. Not only is the software first-rate, but I can also make an infinite number of back-up copies, even give them to my friends, but only one copy will run at a time! They also make the best 80 column adapter there is for the Commodore 64! THANK YOU Batteries Included.

K.J. Rogerson, Farmington Hills, MI

Thanks for the tremendous vote of confidence. It's nice when a reader takes the time to respond to matters we expose each issue. Also, Batteries is sure to feel pleased when they read of the respect you hold for their policies and product line. I agree. Their products are fine and the dongle protection scheme is top class. If only a few more software manufacturers felt the same as the public seems to. Although disk protection is said to be used to keep prices down, all it usually accomplishes is to raise the red flag for the pirates, and to keep the legitimate purchasers ticked off at the inconvenience.

A clearinghouse is ideal theory, but implementing such a service would require a lot of legwork, and it seems there are not enough seconds in a minute as it is. Any volunteers?

More Ad-vice: I was on the brink of writing to you to ask why you don't include more advertising when in came the Nov 85 issue with John Brunner's letter on the same topic.

I have recommended Transactor to several friends, some of whom have responded that they don't like the fact that it has no advertising! I like to see some too, especially all the little ones (things like cheat sheets for example, and specialized software from little known sources).

It is a good idea to keep it all in one place, and I favour the middle. It can then be removed without harming the magazine when the file has to be thinned out maybe six months or a year after receipt. Wherever it is, it should be bound in such a manner that the advertising pages can be removed without releasing the editorial pages. I guess seven pages is a good number if it includes the back cover.

If you want to keep advertising down, but are in a position to accept ads, you have the best of both worlds. You can set the rates at a level which attracts only the number of ads you want to accommodate. Set the rates deliberately high in the first place, and if you don't get enough advertisers, reduce the rate until you achieve the desired amount. And if you are accepting ads with a certain degree of reluctance, you don't have to worry about the old adage about who pays the piper calls the tune. It doesn't apply if the piper has an independent source of income.

R.C. Eldridge Pemberton, British Columbia

Since we published John Brunner's letter of advice, we have been getting a number of really choice letters to back up Brunner's philosophy. Bob, thanks for the great advice. You may be right about the middle section of the magazine. Although it would mess up the nice flow we try so hard to attain, it would have the advantage of being quite easily removed when no longer required. Super idea.

As for your idea of setting rates as high as possible, then adjusting to get just the right amount of advertising, another bright idea. How does \$5000.00 per page sound to start with? Although our print run is far below many of our competitors, we do cater to a more select crowd of people. We feel that our readers are the ones that write the software, design the hardware, and pull the trigger of the industry, as it relates to Commodore. How's that for cheeky. An advertiser with the right product would get a much better response with us than with our competition. The problem is that advertisers with quality products like to have their ads placed alongside similar products to allow comparison by the readers. Although we could sift out all but the ads for quality products, it might mean discriminating between two equally "paying customers" which isn't fair. But we're working on it! Thanks again.

CP/M In The Transactor?: I have been very pleased with the technical nature of your magazine. I only wish I could buy 12 issues a year. Keep up the good work. I have a few thoughts that keep running through my head that I'd like to pass on to you.

First, I was wondering now that I've bought a C128, do you intend to deal with the CP/M in the same way as CBM topics or would we be better advised to subscribe to a CP/M oriented magazine such as Dr. Dobbs's or Language? I ask this because CP/M is almost an entirely new world, as is the Amiga, and to a lesser extent the C128. I have serious reservations that justice can be done to all this new material.

Second, I would like to respond to your intent to marginally increase the advertising content of your magazine. I, and I suspect many others, have mixed feelings on the subject. I think that advertisements can be an important source of information to the reader (I will, on occasion, buy a Byte just for a listing of available products on the market). Advertisers are obviously an important source of revenue to your magazine. I think that for your own growth as a competitive magazine you cannot ignore the necessity of incorporating more advertisements. Unfortunately, advertisements traditionally tend to provide little substance to a magazines content. I would like to suggest some possible remedies of a somewhat radical nature:

I – Enforce advertising standards. (I don't know if this has ever been done?) Insist that advertisers include user information of a technical nature. Make advertisements a coordinated segment of your magazine. I would like to see cost, content, comparisons, and more. If advertisements provided all of that then maybe people would be more accepting of them. (Can you tell advertisers what to do?)

II – Allow advertisements only on a annual basis (or longer) and charge more. This would require an index of some nature. This index could be cross-referenced by company and subject with advertisers paying for extra index entries. You could even expand the index to include references to Transactor articles. (Am I getting carried away here?) In any case, under such a plan, the information content goes up and the number of ads goes down.

III – Institute reader service cards. Incomplete information is often more irritating than no information. User cards put complete information within easy reach and, in my view, make advertisements much more palatable. I know that one of the primary reasons I am a computer nut is because I am an information nut. The more quality information you make available, the more I like you.

By incorporating these concepts, all parties should be happy. Readers have a better quality advertisement and an index to allow comparison shopping when they are interested in it. Transactor can keep down the space required for advertisements while at the same time increasing their advertising base. Advertisers are insured that readers who are interested in their product have easy access to their information for an extended period of time via the index. This format would also encourage advertisers to place ads in issues whose content relates to their product.

Finally, have you seen Info Magazine's method of providing disk versions of their listings? Rather than selling by issue, they let you choose what you want and you get charged by the number of disks sent. I find myself not ordering disks because I have little interest in half of what's on the disk. I would recommend that you consider this method. It is an exceptionally well conceived idea, and it would fit in well with an already well put together magazine.

Eric B. Wolff, Cinti, OH

We do not intend to cover the CP/M option of the C128 to any great extent. Many of the other magazines have been operating in CP/M for years now, so it would be difficult for us to break in so late and expect to compete. Covering the Amiga is another story, though. The Amiga appears to be a fresh, new entry in a market stagnating with compatibles and clones, and as such deserves a little press ala The Transactor. As a matter of fact, just a few days ago a friend of mine, Jesse Knight, called from Texas to offer his services in the Amiga writing department. Jesse has an Amiga development system and is so impressed with it that he can't wait to tell the world. We should have our own Amiga pretty soon and The T will be firmly on the receiving end of our own Amiga discoveries.

About your advertising comments, not too bad. We appreciate the time and effort you must have spent in composing your letter.

You have probably read the answer to our previous letter by now. Once again, we're working on it, and thanks again for the suggestions. As for the index, you probably have not seen some of the earlier Transactors. Each had an Advertising Index that looked much like what you describe, but quite frankly, they didn't leave much of an impact with our readers or advertisers. Or am I wrong here?

Making additional technical info a mandatory advertising requirement would be terrific for the readers. Advertising that actually lasts longer than the turn of a page. A sharp concept. But it would be difficult to enforce and we might end up right where we are - no advertisers. Indeed there are many aspects to examine.

Including a reader service card would be a natural for us if we start running ads again. Not only would it appeal to the advertisers, but it would also provide a better service to our readers. Our ads won't come cheap, but then quality goods usually don't.

The last point mentioned regarding Info Magazine's disk service is hard for me to comprehend. We deal in such a vast number of diskettes every issue that a custom selection would be next to impossible. For us, the idea would not be feasible even with reduced sales. Our staff is just not large enough to provide such a service plus get a magazine out. Sorry to say that the idea sounds great for the consumer but would be difficult to

provide from our viewpoint. We're considering making our programs available for downloading from a timesharing service. However, downloading just two or three programs from one disk might cost just as much as the disk itself. Downloading three programs from three different disks might save you money, but the service may not have space for more than one of our Transactor disks at a time.

As a final note, I would like to thank you for your ideas. Through advice like yours will be built a magazine that will appeal to everyone associated with it. Thanks once again.

Medium Interest: If the magazine appears on microfiche, I will immediately switch my subscription to that medium. I will also order as many back copies as I can afford. There are certainly many others in your readership who will do likewise.

Jim Alix, North Vancouver, British Columbia

Alright! Another vote for microfiche. That adds up to two so far. For everyone reading who is so inclined, the microfiche question remains unresolved. We need your letters to qualify our decision to provide microfiche service. Thanks for taking the time to write.

SX Effects: I presently own a 64-SX computer with a single disk drive. I would like to add one more disk drive to the computer but can't get an answer from Commodore. Any help you could give me would be appreciated.

Tim Fucile, Thunder Bay, Ontario

To get the answer to your question, I went to the local SX64 expert, Jeff Goebel. His answer was yes, you can add an extra drive or two to the SX64. Of course if you add a drive that has a device number of 8, you'll have two device 8's. Not good. So the trick is to change the unit address of the internal drive first, say, to #9:

```
open1,8,15  
print#1, "m-w" chr$(119)chr$(0)chr$(2)chr$(9+32)chr$(9+64)
```

Then plug in the external drive and use it as device #8.

If you want the internal drive to be #8 and the external drive #9, you'll have to do some juggling. First you'll have to send the internal drive number "to a neutral corner". Change it to, say, #15 by replacing the two 9+ in the above command string with 15+. Then plug in the external drive #8, change it to #9, and change #15 back to #8. You'll have to open two command channels, one for each device.

The reason for the computerized gymnastics is because you cannot turn off the internal drive separate from the SX64. When you connect an external drive that is still wired for device #8, you can't change the device number of one without changing

them both. A separate power switch for the drive would have been ideal.

We get an awful lot of letters in our mailbox every working day and can imagine Commodore getting several times more. Answers often take time and research. To everybody who has ever sent us a letter but has not received a reply, please do not take it personally. It gives us great pleasure to get such reader response. However, economizing on answers is all too often necessary. It means we have to select an intersection set of questions. Commodore is no doubt bound by the same economics. Perhaps getting less questions than Commodore is one reason we have time for the occasional exception.

Bloops Blues: Not being sure if it's misprints or me, I am writing for help regarding a couple of programs in the Volume 5, Issue 04 of Transactor. (I think that's right, but not sure as on page 1 upper left in large letters it says issue 5. On bottom right of all the pages it says issue 4)

To help pinpoint the issue, it starts with a Hildon article "We're so Misunderstood!"

First problem dealt with program by Rick Illes in the Bits & Pieces section. After entering program name and type, disk runs (red light on) then screen goes to READY, keyboard locks up and drive led flashes. I have a C64 so entered the 4 changed lines 110, 120, 130, 300. Even tried it after changing 130 to read `..*peek(56):s=t` & still no go. (yes, have proof read the entries against the printed copy)

Next comes "Learning the Languages of DOS" by Evers. On page 50, he writes "...check article 'Drive Peeker' in this issue." Well, I've looked and looked and didn't find any article titled 'Drive Peeker'; am I going blind?

Further, I typed in memory read program. It will not accept any \$0000 or any number with an alpha character in it, as a start or end address. So I entered a number like 0300; however, the screen listing didn't start at 0300, the same was true no matter what starting number I used.

I will say though that I found all the articles to be informative and well written.

N.J. Schoenmaker, Ludington, Michigan

To clarify, the title of the issue in question is 'Hardware and Peripherals', Volume 5, Issue 05, dated March 1985 but actually released around December 1984. The designation of Issue 04 on the bottom of each page was caused by a minor slip during typesetting. We forgot to update the border routine to reflect the correct Volume and Issue numbers. We had hoped nobody would notice.

Rick Illes Single Disk Copy Program will work by changing line 130 to read:

`130 t=peek(55)+256*peek(56):s=t`

As you specified, location 46 was referenced instead of 56. But after this change, the program works fine as printed.

My 'Drive Peeker' article did make it to the type shop for that issue, but was not printed due to priorities. We had such a multitude of great articles that Drive Peeker got the boot. The sad part is that my 'Learning ...' article was not updated to reflect this last minute cut. Drive Peeker did appear in the issue following, Volume 5 Issue 06, Programming Aids and Utilities, on page 71.

The Memory-Read demonstration program in the 'Learning The Language of DOS' article does work as listed. When you run the program, though, you must enter the start and ending address to view in decimal, not hex. The routine will display the disk contents in Hex and Ascii, along with showing the current location within DOS RAM or ROM in 16 bit hex at the beginning of each eight character line. The program works, but could have used a bit more explanation I suppose.

Thanks for the final vote of confidence at the end of your letter. It's letters like yours that keep us on our toes.

SuperPet Switch Glitch: I have a few questions that I hope either you or your readers will be able to help me with. I recently purchased a Pet system. It consists of a Super Pet (SP9000), D9090 and 2031 Disk Drives, and a 8023P Printer. My first question regards the two three position switches located on the right side of the Super Pet. I have found that the one switch, marked 'Prog-6502-6809' selects from Waterloo to CBM Basic. The second switch seems to make no difference in operation of the computer regardless of its position, it is marked 'Prog-R/W-Read', what do these two switches really do?

As you may have guessed, I do not have a SP9000 manual. I received a 'Series 8000' manual. Is this the correct manual or is there a SP9000 manual and should I try to get one? The only software that I received was 'The Manager' with its two manuals and the required 'dongle' to run it. I would greatly like to get hold of the 'Waterloo' language programs for the SP9000. I have all the manuals, Basic 6809, assembler, micro apl, micro pascal, and micro fortran, but no programs. Any information as to where I could get these programs would be greatly appreciated.

With the help of a friend I have found out how to run my C64 basic programs on the SP9000, so I found some uses for the SP9000, but not the one I really want.

C. Daniel Schein Jr.
2455 McKinley Avenue
West Lawn, PA 19609

The SP9000 has 32k of user RAM at \$0000-\$7FFF, screen RAM at \$8000-\$8FFF, plus an extra 64k of Bank Switched RAM at location \$9000, accessed in 16-4k blocks. In 6809 mode, with an interpreter Loaded and executing, this RAM is not to be used. The Interpreter lives in it. But, if working from 6809 assembler, or in any mode with the 6502 processor, this area is open for whatever use you may have.

There are two memory locations that are important for use when programming the SP9000. They are:

\$EEF8, the System Latch
\$EEFC, the Bank-Select Latch

The bit configurations are listed below.

\$EEF8 System Latch

Bit Description

- 0 CPU Select: 0 = Motorola 6809, 1 = MOS 6502
- 1 Memory Protect: 0 = Read Only, 1 = Read/Write
- 2 Unused
- 3 Diagnostic Sense
- 4-7 Unused

If the toggle switches are not set to their 'Prog.' positions, all work with the System Latch will be ignored. Bit 3 should be set before switching into MOS 6502 mode. This has been implemented due to the destructive reset sequence the Pet follows on reset. When the Diagnostic Sense is set, this reset sequence is not followed, thereby retaining RAM in transit.

\$EEFC Bank-Select Latch

Bit Description

- 0-3 RAM Bank Number In Use: 0-15
- 4-6 Unused
- 7 Must be set (1) when accessing the System Latch, otherwise should be cleared (0).

The two three position switches perform the following functions.

Switch #1

- Read When set, is the Read Only position of the bank switched RAM at \$9000.
- R/W When set, allows Reading from and Writing to the RAM at \$9000.
- Prog. Software control of the write protect state of the RAM at \$9000 (see System Latch).

Switch #2

- 6809 The Motorola 6809 and Waterloo operating system is provided in this position.
- 6502 The MOS 6502 and Commodore operating system is provided in this position.
- Prog. Selection of the processor in use through software control (see System Latch).

The manuals supplied with the SP9000 were as follows:

The Commodore Business Computer User's Guide Series 8000
The Commodore SuperPET System Overview
The Commodore SuperPET Waterloo 6809 Assembler
The Commodore SuperPET Waterloo microAPL
The Commodore SuperPET Waterloo microBASIC
The Commodore SuperPET Waterloo microCOBOL
The Commodore SuperPET Waterloo microFORTRAN
The Commodore SuperPET Waterloo microPASCAL

The last time I looked, SAM's books carried all of the SuperPET manuals listed above.

Two diskettes were supplied with the SP9000:

1. "The Commodore WCS", which contains the Waterloo APL, Basic, Cobol, Fortran, and Pascal interpreted languages, the 6809 Assembler/Editor system, plus the Waterloo library routines, and
2. The Commodore Tutorial diskette, which contains all of the tutorial examples referenced by the various software manuals supplied.

Both diskettes were once a standard stock item for Commodore here in Canada, but that was a few years ago. You could try to contact Commodore up our way for help, or maybe even Commodore down in West Chester. You never know. Commodore may have some old stock still kicking about. Just in case they don't, I have included your complete address at the bottom of your letter. Hopefully, a kind reader will contact you to offer a few suggestions. If all else fails, you could contact The University of Waterloo in Waterloo, Ontario, Canada. Although rumour has it that they don't support the SuperPET any longer, they won't bite you for trying.

Please remember, even if you don't succeed in your quest for SuperPET information, the machine still makes a great computer. As explained earlier in this column, the 40 and 80 column Pets had quite a selection of top notch software on the market before the advent of the 64. Just find a copy of the Commodore Software Encyclopedia (SAM's Books), and go shopping. Between the software listed in this book, plus the great scads of software available through many of the older Pet users groups, there should be no trouble in locating plenty of good software. Best of luck.

TransBASIC Installment #7

Nick Sullivan
Scarborough, Ont.

TransBASIC Parts 1 to 3 Summary:

Part 1: *The concept of TransBASIC – a custom command utility that allows one to choose from a library only those commands that are necessary for a particular task.*

Part 2: *The structure of a TransBASIC module – each TransBASIC module follows a format designed to make them simple to create and “mergeable” with other modules.*

Part 3: *ROM routines used by TransBASIC – many modules make use of ROM routines buried inside the Commodore 64. Part 3 explains how to use these routines when creating new modules.*

Part 4: *Using Numeric Expressions – details on how to make use of the evaluate expression ROM routine.*

Part 5: *Assembler Compatibility – TransBASIC modules are written in PAL Assembler format. Techniques for porting them to another assembler were discussed here.*

Part 6: *The USE Command – The command ‘ADD’ merges TransBASIC modules into text space. However, as more modules are ADDED, merging gets slow. The USE command was written to speed things up. USE also counts the number of modules USED and updates line 95 automatically.*

To take advantage of the TransBASIC command system, one must first obtain a copy of the TransBASIC Kernel. The Kernel is only about 500 bytes long, but the source listing of the Kernel is quite long and can't be printed each time. Volume 5, Issue 05 (Hardware & Peripherals) contains the printed listing, however The Transactor Disk for every issue will include this file, plus files from the current and all previous TransBASIC articles.

TransBasic Part 7

If you have been following TransBASIC for the last several issues, you will be relieved to learn that this installment is rather shorter than the two that preceded it. This is due to a lack of time, however, not to a lack of fresh modules, which are in plentiful supply.

Darren Spruyt is back this issue with a set of three editing commands. Included are a block delete, automatic line numbering, and line renumbering. The AUTO command in Darren's module works by intercepting the BASIC warm start vector. This marks the first time that TransBASIC has altered

any system vector apart from those used by the kernel, and it introduces the possibility of a new problem that we will have to deal with.

The problem will arise when more than one module changes the same vector. When you build a TransBASIC dialect, the various modules you merge into the source code have no knowledge of each other and, if they happen to want to use a vector, there is no way for them to take into account the possibility that other modules might also be claiming that vector for their own purposes. It is not very likely that the resulting conflict would be peacefully resolved, and a crash would probably ensue.

I suppose it would be possible to graft some supervisory routine onto the kernel even at this late date, which could arbitrate access to the various vectors in some reasonable way. However, it seems better to me to deal with vector conflicts manually. To make this easy requires only that we establish a couple of rules to be followed in writing modules that alter vectors.

The first rule is that the routine to initialize the vector should be called from a line within a special area of source code labelled 'MORVEC'; and that the routine to restore the vector should be called from a another area labelled 'KILVEC'. MORVEC is permanently assigned lines 9162 through 9180; KILVEC gets the next twenty lines, from 9182 to 9200. The code in these areas is executed with JSR MORVEC on line 2125, and JSR KILVEC on line 2555. These instructions must also be included in any module that alters vectors.

The second rule requires that if a particular vector has been used by an existing module, the lines assigned to it in MORVEC and KILVEC are also to be used in any future module that alters the same vector. This will mean that if both modules occur in the same dialect, only one will function, since the second one to be merged will overwrite the initialization call of the first. This might be a nasty surprise, but it should avoid crashes.

By the way, a thorough discussion of vector conflicts, and of what can be done about them, can be found in Chris Zamara's article, 'Flexible Vector Management', in Volume 6, Issue 2.

Another module in this issue comes from Anton Treuenfels, who has provided versions of RESTORE, GOTO and GOSUB with computed destination lines. Anton's own version of this module used the keywords CRESTORE, CGOTO and CGOSUB. However, the native BASIC command is in each instance a subset of the computed command, so I dropped the Cs for the version published here. Decoding the line numbers via the

expression evaluator, as this module does, is going to be a hair slower than the conversion routine used by the ROM, but this won't often matter. If you would prefer to have the ROM commands co-exist with Anton's computed commands, change the keywords back. (This is not necessary with RESTORE, of course, since the ROM version won't accept a line number anyway.)

The other modules in this issue provide a couple of commands borrowed from LISP via LOGO (the FIRST\$(and BF\$(functions), and a command to return a random integer in a specified range.

Last issue The Transactor published a very handy unassembler for the C-64, by J. Lothian of Ottawa. Unfortunately for PAL users, the program generates sequential files of assembly language source code, without numbers. Such files are used by the Commodore assembler, but are not compatible with PAL. Converting from one format to the other is not arduous, however. The short BASIC program given as Program 5 will take care of converting the sequential file to a numbered program in memory. There will still be some editing needed to make the file fully PAL-compatible — as a start, you'll need to delete the conversion routine itself, and put a SYS 700 line at the top of the file.

Next issue, look for a multicolour graphics package from Darren Spruyt, a modified INPUT or two, and more.

New Commands

AUTO (Type: Statement Cat #: 136)

Line Range: 9406–9480

Module: PRG MANAGEMENT

Example: AUTO 100,10

Begin automatic line numbering from line 100, with increments of 10. This command intercepts the BASIC warm start vector at \$0302. It will not be compatible with any future TransBASIC commands that use this vector.

DEL (Type: Statement Cat #: 137)

Line Range: 9234–9404

Module: PRG MANAGEMENT

Example: DEL 275–890

Example: DEL –

Delete specified lines. The line number syntax is the same as the LIST command, except that DEL by itself does nothing. The second example is equivalent to NEW — be careful.

REN (Type: Statement Cat #: 138)

Line Range: 9482–10096

Module: PRG MANAGEMENT

Example: REN 300–600,20,5

Example: REN 10,1

Renummer a range of lines with a specified new starting line number and increment. Statements with target line numbers (GOTO, GOSUB, ON–GOTO, etc.) are modified accordingly, with the exception of LIST and RUN. If no line range is given, as in the second example, the whole program is renumbered.

RESTORE (Type: Statement Cat #: 139)

Line Range: 10124–10148

Module: COMPUTED CMDS

Example: RESTORE

Example: RESTORE 1200

Example: RESTORE N + 100

Identical to BASIC's RESTORE statement except that, as in the second example, a line number may be given specifying a point in the program at which the next data read is to begin. The line number is evaluated as a BASIC expression, hence the third example is legal.

GOSUB (Type: Statement Cat #: 140)

Line Range: 10150–10180

Module: COMPUTED CMDS

Example: GOSUB SQR(PX + C%(6))*RND(1)/SIN(1.2)

This is a computed GOSUB. The target line is determined by evaluating the expression after the GOSUB keyword.

GOTO (Type: Statement Cat #: 141)

Line Range: 10182–10188

Module: COMPUTED CMDS

Example: GOTO 100

Example: GOTO BLAZES

This is a computed GOTO. In the second example, the target line is the determined by evaluating the variable BLAZES.

RANDOM((Type: Function Cat #: 027)

Line Range: 3216–3268

Module: RANDOM

Example: FOR I = 1 TO 220: PRINT RANDOM(100,999);
: NEXT

A random integer is returned in the specified range (inclusive). The range boundaries are treated as integers.

FIRST\$((Type: Function Cat #: 028)

Line Range: 3330–3452

Module: PHRASE SPLITTERS

Example: PRINT FIRST\$(" TRIALS, TROUBLES,
TRIBULATIONS")

The first word in the string argument is returned, up to but not including the first space or shifted space. Leading blanks are ignored. If the string is null or consists only of blanks, a null string is returned.

BF\$((Type: Function Cat #: 029)

Line Range: 3334–3452

Module: PHRASE SPLITTERS

Example: PRINT BF\$(BF\$(BF\$(A\$)))

The argument string is returned beginning with the space or shifted space after the first word. BF stands for 'but first', after a LOGO primitive with a similar purpose. If the argument string is null, or consists only of blanks, or has only one word and no trailing blanks, a null string is returned.

Program 1: PRG MANAGEMENT

```

NG 0 rem prg management (d. spruyt 1985) :
FH 1 :
GH 2 rem 3 statements, 0 functions
HH 3 :
DJ 4 rem keyword chars: 10
JH 5 :
NJ 6 rem keyword      routine      line      ser #
FD 7 rem s/auto       car          9406     136
FD 8 rem s/del        dlt          9234     137
OE 9 rem s/ren        rnum         9482     138
OH 10 :
PJ 11 =====
AI 12 :
FK 139 .asc "autOdeLreN"
HA 1139 .word car-1,dlt-1,rnum-1
IC 2125      jsr  morvec
EL 2555      jsr  kilvec
OI 2620 usfp  ldx  #0
AJ 2622      stx  $0d
MA 2624      sta  $62
BH 2626      sty  $63
ON 2628      ldx  #$90
HM 2630      sec
NH 2632      jmp  $bc49
AM 2634 ;
EE 9150 errpgm ldx  $3a      ;test hi byte of
EJ 9152      inx            ;'curlin' = $ff
GI 9154      bne  epg1      ;no
AL 9156      rts
KI 9158 epg1  jmp  $af08     ;'syntax'
OD 9160 ;
NF 9162 morvec = *
NF 9164      jsr  auton      ;warm start $302
GM 9178      rts
CF 9180 ;
MJ 9182 kilvec = *
ID 9184      jsr  autoff     ;warm start $302
KN 9198      rts
GG 9200 ;
FN 9202 autoff lda  wrmsrt
MH 9204      sta  $302
IE 9206      lda  wrmsrt + 1
EI 9208      sta  $303
GO 9210      rts
CH 9212 ;
HK 9214 auton lda  $302
OM 9216      sta  wrmsrt
AF 9218      lda  $303
EJ 9220      sta  wrmsrt + 1
OK 9222      lda  #>auto
EJ 9224      sta  $303
GL 9226      lda  #<auto
EJ 9228      sta  $302
KP 9230      rts
GI 9232 ;
HB 9234 dlt   jsr  errpgm
LL 9236      jsr  $a96b      ;get fixed pt number
EK 9238      ldx  #$50

```

```

AC 9240      jsr  store      ;save it
KA 9242      cmp  #$ab      ;" - " token value
OD 9244      bne  dlt2
OB 9246      jsr  $73
OD 9248      bne  dlt1
KL 9250      lda  #$ff      ;if no top, then
OA 9252      sta  $52      ;set upper
MI 9254      sta  $53      ;limit to $ffff
CF 9256      bne  dlt4
DC 9258 dlt1  jsr  $a96b     ;get fixed pt num
LJ 9260 dlt2  sec
JM 9262      lda  $14      ;move top line
FF 9264      sta  $52      ;num to ($52)
DL 9266      sbc  $50      ;and check that
GF 9268      lda  $15      ;num2 >= num1
DA 9270      sta  $53
BN 9272      sbc  $51
HH 9274      bcs  dlt3
JD 9276      jmp  $af08     ;'syntax'
PH 9278 dlt3  inc  $52      ;bump num2
KG 9280      bne  dlt4
BP 9282      inc  $53
CM 9284 dlt4  ldx  #$50
OD 9286      jsr  uns      ;num1 to ($14)
OH 9288      jsr  $a613     ;find line
DG 9290      lda  $5f      ;copy pntr
IK 9292      sta  $22      ;to ($22)
GN 9294      lda  $60
HB 9296      sta  $23
II 9298      ldx  #$52     ;num2 to ($14)
DB 9300      jsr  uns
MI 9302      jsr  $a613     ;find line
GO 9304      jmp  dlt11     ;skip over seg
MM 9306 dlt5  sec
AB 9308      lda  $2d      ;move from
FA 9310      sbc  $5f      ;($5f)-($2d)
CF 9312      sta  t2      ;to ($22) ↑
AN 9314      lda  $2e      ;calc pages
OJ 9316      sbc  $60      ;+ remainder $02
BD 9318      tax
EE 9320      inx
FL 9322      ldy  #0
NE 9324      beq  dlt7      ;init .y
DN 9326 dlt6  lda  ($5f),y  ;transfer data
AH 9328      sta  ($22),y
CF 9330      iny
GK 9332      bne  dlt6
OB 9334      inc  $60
BC 9336      inc  $23
LC 9338 dlt7  dex          ;more pages
GL 9340      bne  dlt6     ;yes
GG 9342 dlt8  cpy  $02
DN 9344      beq  dlt9
GK 9346      lda  ($5f),y  ;copy remainder
EI 9348      sta  ($22),y
GG 9350      iny
CM 9352      bne  dlt8
NJ 9354 dlt9  tya          ;calculate
DD 9356      clc          ;new start-of-
AC 9358      adc  $22      ;variables pntr

```


KI	9360	sta \$2d	
HL	9362	bcc dlt10	
ND	9364	inc \$23	
CM	9366 dlt10	lda \$23	
FJ	9368	sta \$2e	
GI	9370	rts	
JB	9372 dlt11	jsr dlt5	;move mem
JC	9374	jsr \$a533	;re-chain
AC	9376	jsr \$a659	;clr/fix pntrs
IJ	9378	jmp \$a474	;return to ready
KB	9380 ;		
LN	9382 store	ldy \$14	;copy from
OG	9384	sty 0,x	; (\$14) to
DB	9386	ldy \$15	; 0,x
DE	9388	sty 1,x	
KJ	9390	rts	
GC	9392 ;		
DL	9394 uns	ldy 0,x	;copy from
IF	9396	sty \$14	; 0,x
FK	9398	ldy 1,x	; to (\$14)
DO	9400	sty \$15	
GK	9402	rts	
CD	9404 ;		
GJ	9406 car	jsr errpgm	;auto function
AI	9408	jsr \$a96b	;get num
NL	9410	jsr aneval	;evaluate expr
HH	9412	sta step	;step value
PE	9414	lda \$51	
OK	9416	ldy \$50	
OM	9418	ldx #1	;set flag
IG	9420	stx cont	
BJ	9422	jmp aut3	;print value
GE	9424 ;		
EE	9426 auto	ldy \$200	;(\$302) vector
IL	9428	bne aut1	;line entered
HP	9430	sty cont	;zero lock
LH	9432 aut1	ldy cont	;get lock
MB	9434	beq aut5	;no go
HE	9436	lda \$14	;take currline#
GF	9438	clc	;and add step
EL	9440	adc step	; value to it
KO	9442	bcc aut2	
BJ	9444	inc \$15	
IL	9446 aut2	tay	
FH	9448	lda \$15	
MJ	9450 aut3	jsr strng	;make string
LB	9452	ldy #\$ff	;push string
ED	9454 aut4	iny	; into kb buffer
MA	9456	lda \$100,y	
PH	9458	sta \$277,y	
NC	9460	bne aut4	
PL	9462	lda #\$20	;add space
FI	9464	sta \$277,y	
KN	9466	iny	
GO	9468	sty \$c6	;set buff chars
EJ	9470 aut5	lda wrmsrt	
KC	9472	sta \$14	;create indirect
EF	9474	lda wrmsrt + 1	
PM	9476	sta \$15	
CL	9478	jmp (\$14)	;jmp thru

OH	9480 ;		
OA	9482 rnum	jsr errpgm	
EI	9484	jsr \$a96b	;eval num
AM	9486	cmp #\$ab	; " - " token
DD	9488	bne rnm3	
CC	9490	ldx #\$45	;save start line
OI	9492	jsr store	
GB	9494	jsr \$73	
CI	9496	cmp #\$2c	;disallow comma
JD	9498	bne rnm2	
MA	9500 rnm1	jmp \$af08	; 'syntax'
EC	9502 rnm2	jsr \$79	
IJ	9504	jsr \$a96b	;eval num
CI	9506	cmp #\$2c	;comma required
PD	9508	bne rnm1	
IO	9510	ldx #5	;save end line
CK	9512	jsr store	
KC	9514	jsr \$73	
EK	9516	jsr \$a96b	;eval num
OI	9518	cmp #\$2c	;comma required
LE	9520	bne rnm1	
OG	9522	beq rnm4	
LJ	9524 rnm3	ldy #0	;if no start/end
IA	9526	sty \$45	; were specified
FF	9528	sty \$46	; renumber whole prg
FP	9530	dey	
HP	9532	dey	
CA	9534	sty t5	
GA	9536	sty t6	
IE	9538 rnm4	jsr aneval	;eval start/step
CC	9540	inc t5	;bump end
BH	9542	bne rnm5	
AJ	9544	inc t6	
NC	9546 rnm5	lda \$45	;(\$45) to (\$14)
EB	9548	sta \$14	
EO	9550	lda \$46	
LB	9552	sta \$15	
II	9554	jsr \$a613	;find line
BE	9556	lda \$5f	;(\$5f) to (\$ac) as
AN	9558	sta \$ac	; pntr to start of
GG	9560	lda \$60	; section to be
NC	9562	sta \$ad	; renumbered
EE	9564	lda t5	;(\$05) to (\$14)
GC	9566	sta \$14	
II	9568	lda t6	
NC	9570	sta \$15	
NM	9572	jsr fndlin	;calc new top line#
DL	9574	ldy #1	
DI	9576	lda (\$5f),y	
PF	9578	beq rnm7	;link zero-okay
MC	9580	ldy #3	;next line hi
JL	9582	lda (\$5f),y	
IL	9584	cmp \$ab	;new top hi
BK	9586	bne rnm6	
PC	9588	dey	
EH	9590	lda (\$5f),y	;next line lo
DN	9592	cmp \$aa	;new top lo
MJ	9594 rnm6	bcs rnm7	;no conflict
IB	9596	jmp \$b248	;illegal quantity
KE	9598 rnm7	lda #0	;clr on/goto/gosub

JD	9600	sta onflg	; flag	GD	9720	jmp \$a474	;'ready.'
GO	9602	lda \$2b	;start-of-basic	NI	9722 rnm16	iny	;next char
AB	9604	sta \$3f	; to (\$3f) trace	PA	9724 rnm17	jsr rnm18	;change l#
PD	9606	lda \$2c		AO	9726	ldy \$97	;restore .y
KE	9608	sta \$40		EK	9728	jmp rnm10	;return
DN	9610 rnm8	ldy #2		IL	9730	;match for change	
OG	9612	lda (\$3f),y	;copy line number	FB	9732 rnm18	jsr nextch	;cut thru space
JG	9614	sta \$39		OC	9734	sty \$97	;save .y
LB	9616	iny	;to (\$39)	NM	9736	clc	;fix chrget
HK	9618	lda (\$3f),y		FG	9738	tya	; to .y plus
HI	9620	sta \$3a		CD	9740	adc \$3f	; (\$3f) - also
BH	9622 rnm9	iny		IK	9742	sta \$7a	; to (\$41)
NK	9624 rnm10	lda (\$3f),y	;get byte	FN	9744	sta \$41	
OI	9626	beq rnm14	;zero byte	GJ	9746	lda \$40	
AM	9628	cmp #\$3a	;colon	BP	9748	adc #0	
CA	9630	beq rnm14		ON	9750	sta \$42	
KI	9632	cmp #\$89	;goto token	GB	9752	sta \$7b	
IA	9634	beq rnm16		IP	9754	jsr \$79	;set flags
CM	9636	cmp #\$8d	;gosub token	FI	9756	jsr \$a96b	;eval number
MA	9638	beq rnm16		PJ	9758	sec	
LF	9640	cmp #\$cb	;go token	IA	9760	lda \$7a	;find length
CE	9642	bne rnm11	; followed by to	AI	9762	sbc \$41	;of
BD	9644	jsr nextch1		CI	9764	sta t2	
HG	9646	cmp #\$a4	;token to	DG	9766	sec	;chk number is
GB	9648	beq rnm16		IC	9768	lda \$14	; in range
NP	9650	bne rnm10		AG	9770	sbc t5	
NP	9652 rnm11	cmp #\$a7	;then token	JL	9772	lda \$15	
DA	9654	bne rnm12		GG	9774	sbc t6	
ND	9656	jsr nextch1		GG	9776	bcs rnm19	;line above top
ED	9658	cmp #\$30	;check for	DL	9778	sec	
GM	9660	bcc rnm10	; numeric	OL	9780	lda \$14	
DH	9662	cmp #\$3a		JN	9782	sbc \$45	
JO	9664	bcc rnm17		FM	9784	lda \$15	
EC	9666	bcs rnm10		AO	9786	sbc \$46	
HK	9668 rnm12	cmp #\$91	;on token	AO	9788	bcs rnm20	;line below bottom
EB	9670	bne rnm13		BM	9790	;line to be renumbered here	
HN	9672	sta onflg	;set on flag	OF	9792 rnm19	jmp rnm31	
OD	9674	jmp rnm9		NB	9794 rnm20	jsr findlin	;figure new l#
AK	9676 rnm13	cmp #\$2c	;commas	JK	9796	ldy #\$ff	;new line at \$100
AC	9678	bne rnm9	;fetch next byte	IA	9798 rnm21	iny	
FF	9680	lda onflg		EG	9800	lda \$100,y	
ON	9682	beq rnm9	;on not active	KJ	9802	bne rnm21	
FL	9684	bne rnm16	;on active	LM	9804	tya	;new length
DF	9686 rnm14	ldx #0		PM	9806	sec	
PE	9688	stx onflg	;clr on state	LA	9808	sbc t2	;sub initial len
OP	9690	cmp #\$3a	;colon	PP	9810	bmi rnm26	;remove some
LG	9692	beq rnm9	;fetch next	LD	9812	beq rnm28	;right size
NH	9694	ldy #1	;check link hi	IL	9814	;add in .a spaces	
FP	9696	lda (\$3f),y		GL	9816	sta t2	
OL	9698	beq rnm15	;end of memory	LN	9818	sec	
PK	9700	tax		MK	9820	lda \$38	;check for space
BK	9702	dey		JB	9822	sbc \$2e	;between end of mem
FA	9704	lda (\$3f),y	;get link lo	HH	9824	bne rnm22	;and start of
MO	9706	sta \$3f		BI	9826	sec	;variables
KA	9708	stx \$40		LP	9828	lda \$37	
DA	9710	jmp rnm8	;start new line	CD	9830	sbc \$2d	
OH	9712 rnm15	ldx #5		NM	9832	cmp t2	
BL	9714	jsr uns		LL	9834	bcs rnm22	;space available
IJ	9716	jsr number	;renumber lines	MG	9836	jmp \$a435	;out of memory
KG	9718	jsr \$a659	;clr	BK	9838 rnm22	lda \$41	;start of block

GH	9840	sta \$5f		HA	9960	;assumed line# is in (\$14)	
IN	9842	lda \$42	;to move (\$5f)	LD	9962	ldy #3	
KD	9844	sta \$60		PO	9964	lda (\$5f),y	;get line hi
FP	9846	lda \$2d	;end of block	DD	9966	cmp \$15	;check it
PG	9848	sta \$5a		OI	9968	bne fdln2	
MG	9850	ldx \$2e	;to move (\$5a)	NK	9970	dex	
CN	9852	stx \$5b		BB	9972	lda (\$5f),y	;get line lo
DF	9854	clc	;end of new block	GA	9974	cmp \$14	;test if found
CJ	9856	adc t2		HJ	9976	beq fdln6	; yes
GP	9858	bcc rnm23	;when moved	BN	9978 fdln2	bcs fdln5	; no
AG	9860	inx		LG	9980	bit numb	;test renum pass
IM	9862 rnm23	stx \$2e	;\$58)	LF	9982	bpl fdln3	;no
CI	9864	sta \$2d		AB	9984	ldy #2	;yes, put
GG	9866	sta \$58		CO	9986	lda \$aa	; new line number
HM	9868	stx \$59		PE	9988	sta (\$5f),y	; into memory
NB	9870	inc \$5a	;inc (\$5a) by 1	KN	9990	lda \$ab	
DO	9872	bne rnm24		IO	9992	iny	
OG	9874	inc \$5b		DG	9994	sta (\$5f),y	
LD	9876 rnm24	inc \$58	;inc (\$58) by 1	DA	9996 fdln3	lda \$aa	;take (\$aa) and
KO	9878	bne rnm25		DG	9998	clc	; add step value
JF	9880	inc \$59		OI	10000	adc \$52	
PE	9882 rnm25	jsr \$a3bf	;move memory	BC	10002	sta \$aa	
GC	9884	jmp rnm28	;put in line #	LI	10004	bcc fdln4	
CO	9886 rnm26	eor #\$ff		KA	10006	inc \$ab	
GB	9888	clc		MO	10008	lda \$ab	
HO	9890	adc #1	;calc # bytes move	IL	10010	cmp #\$fa	;test over 64000
CA	9892	sta t2		PC	10012	bcc fdln4	;no
JP	9894	lda \$41	;\$22) = bottom	FJ	10014	jmp \$b248	;'illegal quantity'
MG	9896	sta \$22		BI	10016 fdln4	ldy #0	
AJ	9898	ldx \$42	;\$5f) = old bottom	KB	10018	lda (\$5f),y	;get new link lo
PM	9900	stx \$23		PO	10020	tax	
AF	9902	clc	;dlt5	IM	10022	iny	;get new link hi
CM	9904	adc t2		DE	10024	lda (\$5f),y	
II	9906	sta \$5f	;move (\$5f)-(\$2d)	DD	10026	bne fdln1	;continue if >0
BO	9908	bcc rnm27		OH	10028 fdln5	lda #\$f9	;drop here if = 0
JB	9910	inx	;to (\$22)-(\$2d)	OB	10030	ldy #\$ff	;set num to 63999
FG	9912 rnm27	stx \$60		GK	10032 ;		
LD	9914	jsr dlt5		MI	10034 strng	jsr usfp	;make fp val
GF	9916 rnm28	ldy #0	;put in linenum	OB	10036	jmp \$bddf	;make strng at \$100
HI	9918 rnm29	lda \$100,y		MA	10038 fdln6	lda t2	;if old len 0 then
IC	9920	beq rnm30		GB	10040	beq fdln5	;make new 63999
EM	9922	sta (\$41),y		OA	10042	lda \$ab	
EK	9924	iny		NG	10044	ldy \$aa	
IH	9926	bne rnm29	;finish up	CN	10046	jmp strng	;make string
LE	9928 rnm30	jsr \$a533	;re-chain	GL	10048 ;		
AL	9930 rnm31	rts	;end	KH	10050 aneval	ldx #\$50	;evaluate two
CE	9932 ;			EK	10052	jsr store	; params & save
FP	9934 fndlin	lda #0	;finds line, calcs	MA	10054	jsr \$73	;used
AK	9936	lda #0	; new line number	BK	10056	jsr \$b79e	; with num
NK	9938	.byte \$2c		MB	10058	stx \$52	; and auto
PC	9940 number	lda #\$80	;also renums	AA	10060	txa	
BB	9942	sta numb	;if entry number	FA	10062	bne anev1	
OF	9944	lda \$50		BF	10064	jmp \$b248	
JO	9946	sta \$aa		DC	10066 anev1	rts	
FG	9948	lda \$51		KM	10068 ;		
PK	9950	sta \$ab	;set start l#	JN	10070 nxtch1	iny	;skip thru
KL	9952	lda \$ad		AG	10072 nextch	lda (\$3f),y	; until
FB	9954	ldx \$ac		FA	10074	cmp #\$20	; non-space
ML	9956 fdln1	stx \$5f		DK	10076	beq nxtch1	
JA	9958	sta \$60	;set start pntr	KE	10078	rts	


```

GN 10080 ;
BE 10082 .if >(&*255) + 7: * = * + (&*1)
KC 10084 wrmsrt.word 0
MN 10086 ;
OL 10088 cont .byte 0
AG 10090 step .byte $0a
LM 10092 onflg .byte 0
DH 10094 numb .byte 0
GO 10096 ;

```

Program 2: COMPUTED CMDS

```

EA 0 rem computed cmds (a. treuenfels) :
FH 1 :
GH 2 rem 3 statements, 0 functions
HH 3 :
NE 4 rem keyword characters: 16
JH 5 :
NJ 6 rem keyword routine line ser #
GG 7 rem s/restore restor 10124 139
HI 8 rem s/gosub cgosub 10150 140
MD 9 rem s/goto cgoto 10182 141
OH 10 :
JD 11 rem =====
AI 12 :
AI 140 .asc "restorEgosuBgotO"
NC 1140 .word restor-1,cgosub-1,cgoto-1
LM 10124 restor jsr $79 ;chk end of stmt
IH 10126 bne crs1 ;no
EG 10128 jmp $a81d ;regular restore
CP 10130 crs1 jsr $ad8a ;get num expr
PG 10132 jsr $b7f7 ;conv to line #
MG 10134 jsr $a613 ;search for line
PO 10136 bcc crs2 ; not found
NA 10138 lda $5f ;line link low
PF 10140 sbc #1 ;find line start
BB 10142 ldy $60 ;line link high
EB 10144 jmp $a824 ;set data ptr
DI 10146 crs2 jmp $a8e3 ;'undef'd stmt'
KB 10148 ;
PD 10150 cgosub lda #3 ;chk stack space
GB 10152 jsr $a3fb
FB 10154 lda $7b ;save chrget ptr
IC 10156 pha
LG 10158 lda $7a
MC 10160 pha
GI 10162 lda $3a ;push crnt line #
AD 10164 pha
DF 10166 lda $39
ED 10168 pha
GM 10170 lda #$8d ;push gosub token
ID 10172 pha
JB 10174 jsr $79 ;get crnt char
NF 10176 jsr cgoto ;get subtrn strt
FO 10178 jmp $a7ae ;execute stmts
KD 10180 ;
KK 10182 cgoto jsr $ad8a ;get num expr
DK 10184 jsr $b7f7 ;conv to line #
DG 10186 jmp $a8a3 ;enter goto rtn
CE 10188 ;

```

Program 3: RANDOM

```

PA 0 rem random (aug 24/84) :
FH 1 :
EC 2 rem 0 statements, 1 function
HH 3 :
HO 4 rem keyword characters: 7
JH 5 :
NJ 6 rem keyword routine line ser #
JE 7 rem f/random( rndm 3216 027
MH 8 :
DI 9 rem u/pshfp1 (3270/063)
HI 10 rem u/pul57 (3308/064)
PH 11 :
KD 12 rem =====
BI 13 :
KM 605 .asc "random"
EO 1605 .word rndm-1
LJ 3216 rndm jsr $ad8a ;eval num1
CE 3218 jsr $bccc ;conv to integer
HD 3220 jsr pshfp1 ;push fac 1
HC 3222 jsr $aefd ;check comma
NA 3224 jsr $aef4 ;eval num2, chk )
KE 3226 jsr $bccc ;conv to integer
DE 3228 jsr $b849 ;add 0.5 twice
MM 3230 jsr $b849
IA 3232 jsr pul57 ;pull num1 to $57
AP 3234 lda #$57
PO 3236 ldy #0
AP 3238 jsr $b850 ;num1-num2
II 3240 jsr $bc2b ;get sign of result
LD 3242 cmp #1 ;test if positive
AN 3244 beq rdm1 ;yes
GL 3246 jsr $bc58 ;abs(fac 1)
FG 3248 jsr $bbc7 ;copy fac 1 to $5c
KD 3250 jsr $e0be ;rnd(1) to fac 1
BJ 3252 lda #$5c ;* (num2 + 1 - num1)
BA 3254 ldy #0
IP 3256 jsr $ba28
EI 3258 lda #$57 ;+ num1
HA 3260 ldy #0
CP 3262 jsr $b867
GF 3264 jmp $bccc ;conv to integer
OA 3266 rdm1 jmp $b248 ;'illegal quantity'
KD 3268 ;
EG 3270 pshfp1 lda #3 ;chk stack space
GD 3272 jsr $a3fb
FC 3274 pla ;save return addr
HJ 3276 sta $71
GF 3278 pla
OJ 3280 sta $72
CH 3282 jsr $bbca ;fac 1 to $57
LB 3284 ldx #0
GC 3286 phf1 lda $57,x ;push $57-$5b
EF 3288 pha
GL 3290 inx
PD 3292 cpx #5
GN 3294 bne phf1
DH 3296 phf2 lda $72 ;restore rts addr
OF 3298 pha

```


BH	3300	lda \$71	
CG	3302	pha	
EN	3304	rts	
AG	3306 ;		
GA	3308 pul57	pla	;save return addr
JL	3310	sta \$71	
IH	3312	pla	
AM	3314	sta \$72	
AI	3316	ldx #4	;pull \$57-\$5a
NA	3318 pl57	pla	
JN	3320	sta \$57,x	
BL	3322	dex	
DA	3324	bpl pl57	
EO	3326	bmi phf2	;branch always
GH	3328 ;		

Program 4: PHRASE SPLITTERS

AO	0 rem phrase splitters (aug 25/84) :			
FH	1 :			
DH	2 rem 0 statements, 2 functions			
HH	3 :			
DE	4 rem keyword characters: 11			
JH	5 :			
NJ	6 rem keyword	routine	line	ser #
NF	7 rem f/first\$(	first	3330	028
NO	8 rem f/bf\$(	bf	3334	029
NH	9 :			
ID	10 rem =====			
PH	11 :			
IF	606 .asc "first\$bf\$"			
JH	1606 .word first-1,bf-1			
CP	3330 first	sec		;flag set = first
AN	3332	.byte \$24		
OB	3334 bf	clc		;flag clr = bf
JO	3336	php		;push flag
FO	3338	jsr \$aef4		;eval str, check)
LD	3340	jsr \$b6a3		;set up ptr to str
MN	3342	sta t3		;save length
PE	3344	txa		;push str data addr
OI	3346	pha		
LM	3348	tya		
CJ	3350	pha		
GF	3352	lda t3		;reserve space in
CG	3354	jsr \$b47d		; string storage
PL	3356	pla		;store data addr
DN	3358	sta \$23		; at (\$22)
IK	3360	pla		
GO	3362	sta \$22		
JJ	3364	ldy #\$ff		;set up .y index,
DM	3366	sty t4		; and flag in t4
GG	3368 bbf1	iny		;bump index
HA	3370 bbf2	cpy t3		;test = str len
HD	3372	beq bbf4		;yes
PE	3374	lda (\$22),y		;test current str
BL	3376	and #\$7f		; byte is space
LG	3378	cmp # " "		; or shift-space
LD	3380	beq bbf3		;yes
HH	3382	lda #0		;clear flag - not

GF	3384	sta t4	; a space
OA	3386	beq bbf1	;branch always
JF	3388 bbf3	bit t4	;test prev char was
IH	3390	bpl bbf4	; space - no
BM	3392	dec t3	;dec effective str
GP	3394	inc \$22	; len, bump
PF	3396	bne bbf2	; effective addr
PO	3398	inc \$23	
LA	3400	bne bbf2	;branch always
OP	3402 bbf4	plp	;test if 'first'
NF	3404	bcs bbf5	;yes
EP	3406	tya	;push index
MM	3408	pha	
BO	3410	sec	;subtract from
EM	3412	sbc t3	; str len
OP	3414	eor #\$ff	
OM	3416	clc	
JD	3418	adc #1	
DB	3420	tay	;use as new index
PP	3422	pla	;add old index to
FI	3424	adc \$22	; address
GC	3426	sta \$22	
NC	3428	bcc bbf5	
PA	3430	inc \$23	
AM	3432 bbf5	tya	;reserve space for
NJ	3434	jsr \$b47d	; new string
LD	3436	tay	
EE	3438 bbf6	dey	;copy string to
NO	3440	cpy #\$ff	; new space
BI	3442	beq bbf7	
GD	3444	lda (\$22),y	
CI	3446	sta (\$62),y	
FE	3448	bcc bbf6	
OE	3450 bbf7	jmp \$b4ca	;create descriptor
CP	3452 ;		

Program 5: Convert SEQ source to PAL source

IG	1 open1,8,2,"0:seqfile":l=1000
CP	2 print "Sq":fori=0to8:printl;l=l+2
IK	3 get#1,a\$:s=-(st=0):ifs=0goto5
NH	4 ifa\$<>chr\$(13)thenprinta\$;goto3
GD	5 print:nexti
EG	6 ifsthenprint "l= ";l;":poke152,1:goto2"
LJ	7 fori=631to639+s:pokei,13:next
CD	8 poke198,9+s:close1-s:print "s";:end
NH	9 :

The Commodore 128: Impressions and Observations

John Holttum
Seattle, WA

Since I have been commissioned by a local software company to translate a book of BASIC 2.0 games to BASIC 7.0, I was able to obtain a C128 and 1571 directly from Commodore during the first week of September (as part of my contract). Now, even with three weeks of programming and evaluation time under my belt, the honeymoon is far from over. As I've told some of my friends, programming in 128 mode evokes feelings close to guilt that so much is so easy! It makes one realize how inhibiting BASIC 2.0 can be. Yet there have also been some disappointments which are worth bringing to light.

First, the machine is not 100% bug-free; not that anyone expected it to be, but one always hopes. So far the problems seem minor. One merely academic bug is in the sequence which brings the keyboard out of auto-insert mode (yes, Virginia, there is auto-insert, and flash, and on-screen underlining). It does not work as stated. That is, if you enable auto-insert with ESC-A, you won't get out of it with ESC-O. However, there is an alternate sequence (ESC-C) which does disable the feature as well as cancelling quote mode.

The second problem is a little more sobering. The SAVE-@ bug is alive and well in the 1571!! I was one of those individuals who had never had too much trouble with SAVE-@ on my 1541, having only been stung by it once even with liberal use of the command. Yet my 1571 has already replaced inappropriate files twice while in 128 mode. Personally, I think the bug is an ancient Gypsy curse placed on CBM and all its descendants, but who's to say?

My final complaint is not a bug at all, but a bit of frustration over the user's manual. As promised, this is the first clearly written and truly attractive book seen from CBM, but I'm repeatedly disappointed by how shallow the material is. Perhaps we C64 hackers are spoiled, having "grown up" with such a well-understood machine. I know that my own history with Apple computers and documentation makes the new 128 manual look more than adequate in content. Still, I am irritated that there is no detailed memory map nor table of vital memory locations (like keyboard buffer, character sets, etc.). Furthermore, the CP/M section offers little practical information, especially for those of us new to the venerable operating system. I asked CBM about that and about the ESC-O problem through CompuServe's CBM Hotline and was informed that the new reference manual was on the way to the printers, whatever that means, but that the Service Manual (part #314001-07) is available by sending \$28.00 to the following address:

Commodore Service Dept.
C-2654
1200 Wilson Drive
West Chester, PA 19380

They knew nothing about the ESC-O bug, but said they would contact the Technical Support Group about it. It may be unique to my machine. So as not to sound too negative, let me say that the BASIC 7.0 encyclopedia contained in the manual is very helpful and well-organized.

So what do I really think of the 128? I think it's undoubtedly Commodore's best effort to date. And despite the above irritations I find it the most satisfying PC I've worked on yet, including the Apple IIe, II+, IIc, C64, and IBM PC. The 128 mode is not just a fat C64. The ESC sequences provide for editing functions far above those for the 64, such as downward scrolling of text, line insertion, partial screen clearing, and easy window formation, all of which are available within

programs as well. The 1571 seems to be able to read anything I put into it, including Kaypro disks, old C64 CP/M disks, and a couple I don't know what they are. RGB output to my \$249 Sears TVS monitor (requires a \$15 cable and includes a TV and composite monitor) is quite nice. It's not the Amiga, but frankly, on my budget, I neither need nor want an Amiga. The 128 system is certainly more computer than I expected to own for many years.

Observations

In the interest of revealing the 128's innards, I have noted three keyboard scanning memory locations I divined by continuously displaying the first 256 bytes of memory on screen and pressing keys to see what happened. I hope they're helpful.

Locations 212 and 213 read most of the keys on the 128 keyboard. The values returned when one of these locations is PEEKed are listed below. These values remain the same whether or not the CONTROL, SHIFT, or CBM keys are also pressed.

A-10	L-42	W-9	7	-24	:	-45	F7/F8	-3
B-28	M-36	X-23	8	-27	;	-50	ESC	-72
C-20	N-39	Y-25	9	-32	,	-47	TAB	-67
D-18	O-38	Z-12	+	-40	.	-44	HELP	-64
E-14	P-41	0-35	-	-43	/	-55	LFEED	-75
F-21	Q-62	1-56	£	-48	=	-53	NO SCROLL	-87
G-26	R-17	2-59	HOME	-51	←	-57	RETURN	-1
H-29	S-13	3-8	DEL	-0	SPACE	-60	NO KEY	-88
I-33	T-22	4-11	↑	-54	F1/F2	-4		
J-34	U-30	5-16	*	-49	F3/F4	-5		
K-37	V-31	6-19	@	-46	F5/F6	-6		

The Keypad on the right has codes separate from the number keys on the main keyboard. They are:

0-81	3-79	6-77	9-78	-	-74
1-71	4-69	7-70	.	-82	ENTER -76
2-68	5-66	8-65	+	73	

Likewise, the cursor keys above the keyboard have different codes from those at the bottom. At the bottom, the LEFT/RIGHT key has a value of 2, while the UP/DOWN key has a 7. At the top of the keyboard, UP = 83, DOWN = 84, LEFT = 85, and RIGHT = 86.

CONTROL, C= (CBM logo), SHIFT, ALT, and CAPS LOCK are all read by location 211. It appears that the first three bits of this byte indicate the status of SHIFT, C=, and CONTROL, respectively, since the value of PEEK(211) is 1 if SHIFT alone is pressed, 2 if C= only is being pressed, and 4 if just CONTROL is pressed. A value of 3 is returned if SHIFT and C= are held simultaneously, 5 for SHIFT and CONTROL, and 7 for all three. Similarly, ALT and CAPS LOCK are represented by bits 4 and 5, respectively. The value of PEEK(211) is 8 when ALT is pressed and 16 when CAPS LOCK is on.

These locations peeked alone or in combination give you control over input of any key on the 128's keyboard, even keys that aren't recognized by the GET or GETKEY commands.

The 128 is not to be underestimated. There is much more to it than at first appears. Happy scanning!

Machine Language: Maxims for the Commodore 128

Jim Butterfield
Toronto

*...the Commodore 128 doesn't call for a new game. . .
...but there are some new things you'll need to know. . .*

The 8500 processor in the Commodore 128 follows the same rules and uses the same instructions as previous Commodore microprocessors – the 6502, 6510, 7501, 6509, etc. There are no new instructions; there's only a new arrangement of pins on the chip (and a new speed capability which can be invoked).

If you know your way around machine language on other Commodore machines, you'll be at home. Same instruction set, same Kernal, similar architecture. But there are some new things you'll need to know.

Here's a set of rules which will keep you out of trouble when you try your hand at programming the Commodore 128 in its C128 mode. All these rules may be broken – when you're smart enough and know your way around. But they will keep you out of trouble when you're just feeling your way around.

1. Use only BANK 15. In fact, you can get along without using the BANK command at all, since bank 15 will be selected if you haven't changed it.
2. Keep program and data below address \$4000. Below this point, there isn't much conflict between banks.
3. If you want to put coding into the cassette buffer, remember it starts at \$0B00. Nobody except the cassette uses that page, so you can use the area from \$0B00 to \$0BFF without worry.
4. Don't try to POKE the screen – the 80 column screen, in particular, is not mapped into main memory. Use the Kernal's CHROUT – \$FFD2 – in the conventional way.
5. The built-in machine language monitor is convenient. It's similar to Supermon. Call it in from Basic with MONITOR, not with some kind of SYS command. You don't need to give a five-character address in most cases when you want to do things such as examine memory, change memory, disassemble, and so on.
6. You'll find the I/O chips at the same addresses as on the Commodore 64, used for 40-column screen, sound, I/O, etc. Feel free to use them in the traditional way, but stay away from (formerly) unused addresses; they may be live and may do unpleasant things to the system.
7. If you need space in zero page for indirect address pointers, use \$FA to \$FF (decimal 250–255).
8. Remember that Basic programs will start at address \$1C01; this may influence where you want to put your program. Remember that Start-of-Variables no longer tells you where your Basic program ends (variables are stored in a different memory bank). Avoid the GRAPHIC command in Basic unless you know what you're doing; it will reorganize memory and move things around.

Remember: you may break any and all of the above rules when you're ready. When you start, using the rules will help you get working programs going.

Simple Example

Let's write an idiotically simple program to print the alphabet on the screen. Go into C128 mode and type MONITOR. Enter the following (the lines will change as you press RETURN, but that shouldn't confuse you):

```
A 0B00 LDX #$41
A 0B02 TXA
A 0B03 JSR $FFD2
A 0B06 INX
A 0B07 CPX #$5B
A 0B09 BNE 0B02
A 0B0B LDA # +13
A 0B0D JMP FFD2
A 0B10 (press RETURN)
```

Disassemble to check (with D 0B00 0B0F). Now ask what the decimal equivalent of \$0B00 is by typing \$0B00. You'll see a value of +2816. Return to Basic with command X.

Ready? Now command SYS 2816 and see the alphabet print.

By the way, exactly the same code will run in all other Commodore machines . . . so the Commodore 128 doesn't call for a new game.

Now . . . when you've learned a little confidence, start experimenting on how to break the above rules.

SUPERMON+64

A new version of the Machine Language Monitor program Supermon has now been released; it's called SUPERMON+. There are versions available for Commodore 64 and for VIC-20.

The new Supermon is designed to closely match the commands of the built-in monitor of the Commodore 128. Users who go from one machine to another (or from C128 side to C64 side within the Commodore 128 computer) might be otherwise confused in trying to remember which commands and which formats need to be used with which monitor.

Some of the useful new features of Supermon+ are: ability to enter decimal or binary numbers anywhere within a command; memory display including ASCII equivalents; number base conversion; and a built-in "wedge" command. On this last feature, command "@" will yield disk status, and command "@,\$0" (note the comma) will call up a directory listing.

Supermon+, like Supermon, is public domain and can be obtained from a variety of sources including TPUG.

COMMODORE 128 Memory Maps: Important Locations

Jim Butterfield, Toronto, Ontario

(Abridged: August 15/85)

These maps apply to the machine when used in 128K mode. In 64 mode, the machine's map is identical to that of the Commodore 64.

There are 28 pages of overhead before the start of Basic. This brief list shows some of the more important locations.

Architecture: "Bank numbers" as used in Basic BANK and the MLM addressing scheme are misleading; in fact, they are more correctly "configuration numbers". Bank 0 shows RAM level 0, which contains work areas and the user's Basic program. Bank 1 also shows RAM, this time (for addresses above hexadecimal 0400) level 1 which contains variables, arrays, and strings. Other "banks" are really configurations, with various types of ROM or I/O overlaying RAM. Thus, bank 15 (the most popular) is ROM and I/O covering RAM bank 0. Bank 14, however, is ROM and the character generator overlaying RAM bank 0. Architecture is set so that addresses below \$0400 reference bank 0 only. Other bank switching (more complex than the simplified 16-bank concept) is accomplished via storing a mask to address \$FF00, or calling up pre-stored masks by writing to \$FF01-FF04.

All Banks:

Hex	Decimal	Description
0000 - 0001	0-1	I/O port, similar to C64
000F	15	Type: FF = string; 00 = numeric
0010	16	Type: 80 = integer; 00 = floating point
0015	21	Current I/O prompt flag
0016 - 0017	22-23	Integer value
002D - 002E	45-46	Pointer: start-of-BASIC (for bank 0)
002F - 0030	47-48	Pointer: start-of-variables (bank 1)
0031 - 0032	49-50	Pointer: start-of-arrays
0033 - 0034	51-52	Pointer: end-of-arrays
0035 - 0036	53-54	Pointer: string-storage (moving down)
0039 - 003A	57-58	Pointer: limit-of-memory (bank 1)
003B - 003C	59-60	Current BASIC line number
003D - 003E	61-62	Textpointer: BASIC work point
0041 - 0042	65-66	Current DATA line number
0043 - 0044	67-68	Current DATA address
0047 - 0048	71-72	Current variable name
0049 - 004A	73-74	Current variable address
0063	99	Accum#1: exponent
0064 - 0067	100-103	Accum#1: mantissa
0068	104	Accum#1: sign
006A - 006F	106-111	Accum#2: exponent, and so on
0070	112	Sign comparison, Acc#1 versus #2
0071	113	Accum#1 lo-order (rounding)
007D - 007E	125-126	BASIC pseudo-stack pointer
0090	144	Status word ST
0091	145	Keyswitch IA: STOP and RVS flags
0098	152	How many open files
0099	153	Input device, normally 0
009A	154	Output CMD device, normally 3
009D	157	I/O messages: 192 = all, 64 = errors, 0 = nil
00A0 - 00A2	160-162	Jiffy Clock HML
00AE - 00AF	174-175	Tape end adds/End of program
00B7	183	Number of characters in file name
00B8	184	Current logical file
00B9	185	Current secondary address
00BA	186	Current device
00BB - 00BC	187-188	Pointer to file name
00C0	192	Tape motor interlock
00C8 - 00CB	200-203	RS-232 input/output buffer addresses
00CC - 00CD	204-205	Keyboard decode pointer (bank 15)
00D0	208	Number of characters in keyboard buffer
00D1	209	Number of programmed chars waiting
00D3	211	Key shift flag: 0 = no shift
00D5	213	Last key code: 88 if no key
00D6	214	Input from screen/from keyboard
00D7	215	40/80 columns: 0 = 40 columns
00D9	217	Character base: 0 = ROM, 4 = RAM
00E0 - 00E1	224-225	Pointer to screen line/cursor
00E2 - 00E3	226-227	Color line pointer
00E4 - 00E7	228-231	Screen margins: bottom, top, left, right
00E8 - 00E9	232-233	Input cursor log (row, column)
00EB	235	Position of cursor on screen line
00EC	236	Row where cursor lives
00FA - 00FF	250-255	UNUSED
0100 - 01FF	256-511	Processor stack area
0100 - 013E	256-318	Tape error log
0100 - 0124	256-292	DOS work area
0125 - 0138	293-312	PRINTUSING work area
0200 - 02A0	512-672	BASIC input buffer
02A2 - 02AE	674-686	Bank peek subroutine
02AF - 02BD	687-701	Bank poke subroutine
02BE - 02CC	702-716	Bank compare subroutine
02CD - 02E2	717-738	JSR to another bank
02E3 - 02FB	739-763	JMP to another bank
02FC - 02FD	764-765	Function execute hook

0300 - 0311	768-785	BASIC links
0312 - 0313	786-787	Unused
0314 - 0315	788-789	IRQ vector
0316 - 0317	790-791	Break interrupt vector
0318 - 0319	792-793	NMI interrupt vector
031A - 032D	794-813	Kernal vectors
032E - 033D	814-829	Kernal links
033E - 0349	830-841	Keyboard matrix shift vectors
034A - 0353	842-851	Keyboard buffer
0354 - 035D	852-861	Tab stop bits
035E - 0361	862-865	Line wrap bits
0362 - 036B	866-875	Logical file table
036C - 0375	876-885	Device number table
0376 - 037F	886-895	Secondary address table
0380 - 039E	896-926	CHRGET subroutine
0386	902	CHRGOT entry
039F - 03D1	927-938	Subroutines to fetch from RAM banks
03DF	991	Accum#1: Overflow
FF00	65280	MMU configuration register
FF01 - FF04	65281-65284	MMU load config registers

Bank 0:

0400 - 07E7	1024-2023	40-column screen memory
07F8 - 07FF	2040-2047	Sprite identity area (text)
0800 - 09FF	2048-2559	BASIC pseudo-stack
0A00 - 0A01	2560-2561	Vector: Basic restart
0A05 - 0A06	2565-2566	Bottom of memory pointer
0A07 - 0A08	2562-2563	Top of memory pointer
0A18	2584	RS-232 receive pointer
0A19	2585	RS-232 input pointer
0A1A	2586	RS-232 transmit pointer
0A1B	2587	RS-232 send pointer
0A20	2592	Keyboard buffer size
0A22	2594	Key repeat: 128 = all, 64 = none
0B00 - 0BBF	2816-3007	Cassette buffer
0C00 - 0DFF	3072-3583	RS-232 input, output buffers
0E00 - 0FFF	3584-4095	System sprites (56-63)
1000 - 10FF	4096-4351	Programmed key lengths and definitions
117A - 117B	4474-4475	Float-fixed vector
117C - 117D	4476-4477	Fixed-float vector
11E9 - 11EA	4585-4586	Light pen values, X and Y
1200 - 1201	4608-4609	Previous Basic line number
1202 - 1203	4610-4611	Pointer: Basic statement for CONT
1204 - 1207	4612-4615	PU characters (.,\$)
1208	4616	Error type ER
1209 - 120A	4617-4618	Error line number EL
1210 - 1211	4624-4625	End of Basic (Bank 0)
1212 - 1213	4626-4627	Basic program limit [FF00]
1218 - 121A	4632-4634	USR program jump
121B - 121F	4635-4639	RND seed value
2000 - 3FFF	8192-16383	Screen memory (hi-res)
4000 - FBFF	16384-64511	BASIC RAM memory (hi-res)

Bank 1:

0400 - FBFF 1024-64511 Basic variables, arrays, strings

Bank 14: Same as Bank 15, below, except:

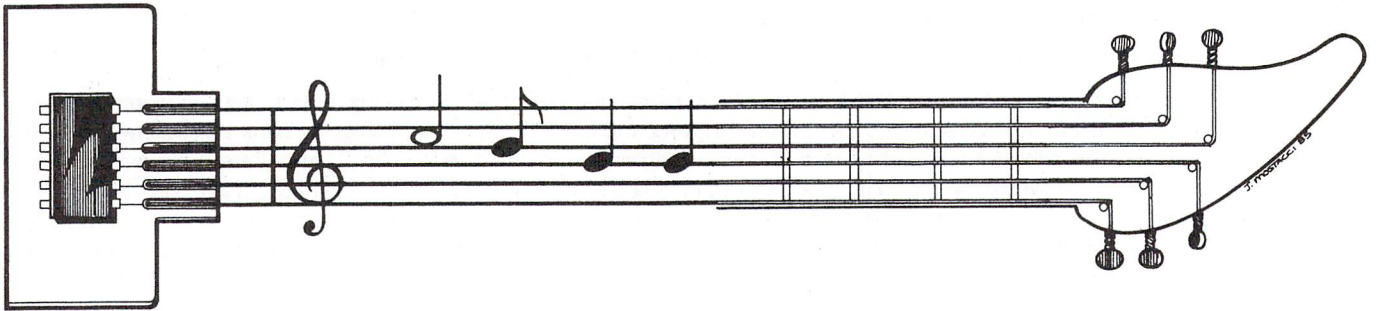
D000 - DFFF 53248-57343 Character generator ROM

Bank 15:

4000 - CFFF	16384-53247	ROM: BASIC
D000 - D030	53248-53296	40-col video chip 8564
D400 - D41C	54272-54300	SID sound chip 6581
D500 - D50A	54528-54538	MMU 8722 memory setup registers
D600 - D601	54784-54785	80-column CRT contr 8563
D800 - D8E7	55296-56295	Color nybbles
DC00 - DC0F	56320-56336	CIA 1 (IRQ) 6526
DD00 - DD0F	56576-56591	CIA 2 (NMI) 6526
DF00 - DF0A	57088-57098	DMA controller
E000 - FEFF	57344-65279	ROM: Kernal
FF05 - FFFF	65285-65535	ROM: Transfer, Jump Table

MIDI – Musical Instrument Digital Interface

Richard Evers, Editor



The idea started slow, but rapidly picked up speed.

Throughout the ages, man has always strived for new and exiting ways in which to amuse himself. Man is the master amuser. One way in which amusement was found, and still is today, is with music. In the beginning, there is a hint that the first musical instruments were rocks, producing a somewhat pleasing sound when smashed together in whatever unison they could muster. Soon, wood was found to possess the enthralling capacity to produce various sounds when struck. As time passed, man developed the art of cutting and forming pieces of wood to produce other truly unique sounds. From this simple beginning, we now have every instrument from the bongo drum to the grand piano. Over thousands of years of patient evolution, man has developed the most beautiful art of all – music.

This unique beginning is a primer to enable you to gain easy entrance into today's world of digital music synthesis, a world far removed from what music has been in past. Although the music synthesizer is capable of cloning the sounds of all man made instruments, it also has the ability to allow or disallow the limitations imposed on the instrument by man's capabilities. With a statement sure to be written off as sheer personal judgement, I will state that the finest of today's music synthesizers are capable of mimicking the sounds of man made instruments to such an extent that the synthetic counterpart will be preferred. But, as with all new concepts, many people will strive to retain the older, more familiar yet less refined methods of past.

Today, in what is known as the age of the microprocessor, the true "State Of The Art" (what does that mean?) phase of our existence, analog systems still persist in great numbers. Although the digital movement has taken all forms of music reproduction by storm, ie. the compact disk, digital recorded record albums and, of course, music synthesis, analog is still with us. Time will be the sole test of our loyalty to a friend long past its prime. Achieving results that were once thought impossible using analog logic now appear common place due the

advent of today's digital high technology. But, justly enough, within a few years travel, our current high tech will also be considered obsolete. The sad fate of time. Obsolescence.

To get on track, a few years ago, when digital synthesis was beginning to reach its apex, a very obvious road block materialized. There were no standards to meet, no industry standard that would allow various components to be easily connected together for use. The manufacturers were producing terrific products, but the buyer was restricted to one name brand for add ons. A restricting situation.

To solve this problem, a few key people in the music synthesis industry started thinking, talking, and attracting attention along this line. The idea started slow, but rapidly picked up speed. MIDI, The Musical Instrument Digital Interface, was the solution presented to combat this problem.

The MIDI system is a series of guidelines that manufacturers should meet in order to ensure the compatibility of their products with others. As with all set guidelines, MIDI is not perfect and does actually impose limitations at times. Without coming down too hard at first, let me explain the guidelines.

The system is based on the almost current 8 bit technology, mainly because the designers were trying to produce a system that would easily fit into the budget of most starving artists. They chose a universal cable connection of the 5 pin DIN plug, using only 3 of the 5 pins. Pin #2 is ground, with pin #'s 4 and 5 being used for a current loop.

" The interface operates at 31.25 kilobaud ($\pm 1\%$), asynchronous, with a start bit, 8 data bits (D0 to D7), and a stop bit. This makes a total of 10 bits for a period of 320 microseconds per serial byte. "

The above paragraph was lifted directly from the MIDI 1.0 Specifications, Document No. MIDI-1.0, Dated August 5, 1983.

With each MIDI equipped unit, you will notice either two or three connectors for MIDI use. There is the MIDI In, MIDI Out, and possibly, MIDI Through (American spelling Thru). MIDI In is the connector for data coming into the unit from other MIDI equipped units on line. This can include drum and rhythm machines, extra keyboards, electric guitar and drum interfaces, controllers and sequencers, or extra synthesizers.

A total of 16 channels, numbered 1 through 16, are allowed under MIDI but more than 16 instruments can be on line at any one time. By assigning duplicate channel numbers to units, or having some units respond to all messages (see Omni Mode), vast numbers of units can be serviced. One main limitation to this provision is the extended line length required, which produces added line loss and a greater time delay with transmitted and received data. By reading more than a few articles on this subject, one fact becomes apparent. When using more than a couple of MIDI units on line, noticeable delays appear in what is deemed simultaneous sound reproduction. But, I have recently read an outstanding interview with a person well qualified in the field. He stated that the time delay between units was not completely the fault of the MIDI implementation. In his opinion, the rate at which the units process the information provided leads to the noticeable time lag. Without greater experience in the subject, it is hard to take a side.

MIDI Out is the data being transmitted from the MIDI equipped unit. MIDI Through is a copy of the information currently being passed into the unit through MIDI In. This feature, if provided, gives the user the ability to slave units together. According to MIDI specs, the cable length between units cannot exceed 50 feet, using shielded, twisted cable with Pin #2 connected to the shielding at both ends.

The transmitted data is quite easy to identify. All transmitted information is 8 bits in length, with the Status bytes having the high bit set and data bytes having the high bit clear. As we will soon discuss, the Status byte has been provided to control the system. Within the Status class will be found all commands necessary to determine what the system will do, when it will

do it, and how it should be done. The Data bytes are really the workers of the system. Not only do they carry information such as the note value to be played, but they also labour under the rule of the Status bytes to ensure that all commands are understood. For the purpose of this article, the eight bit byte is arranged from bits 7 to 0, with bit 7 acting as the high bit.

The charts below have been prepared to show all the possible byte values encountered while working with the MIDI system. Complete explanations will follow.

Table Of Notes Corresponding To Data Values
Expressed In Hex

		Notes											
Octave	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	
-	\$00	\$01	\$02	\$03	\$04	\$05	\$06	\$07	\$08	\$09	\$0A	\$0B	
0	\$0C	\$0D	\$0E	\$0F	\$10	\$11	\$12	\$13	\$14	\$15	\$16	\$17	
1	\$18	\$19	\$1A	\$1B	\$1C	\$1D	\$1E	\$1F	\$20	\$21	\$22	\$23	
2	\$24	\$25	\$26	\$27	\$28	\$29	\$2A	\$2B	\$2C	\$2D	\$2E	\$2F	
3	\$30	\$31	\$32	\$33	\$34	\$35	\$36	\$37	\$38	\$39	\$3A	\$3B	
4	\$3C	\$3D	\$3E	\$3F	\$40	\$41	\$42	\$43	\$44	\$45	\$46	\$47	
5	\$48	\$49	\$4A	\$4B	\$4C	\$4D	\$4E	\$4F	\$50	\$51	\$52	\$53	
6	\$54	\$55	\$56	\$57	\$58	\$59	\$5A	\$5B	\$5C	\$5D	\$5E	\$5F	
7	\$60	\$61	\$62	\$63	\$64	\$65	\$66	\$67	\$68	\$69	\$6A	\$6B	
8	\$6C	\$6D	\$6E	\$6F	\$70	\$71	\$72	\$73	\$74	\$75	\$76	\$77	
9	\$78	\$79	\$7A	\$7B	\$7C	\$7D	\$7E	\$7F					

Status Bytes With Messages

Bit Pattern	Decimal Value	Hex Value	Data Bytes	Status Byte	Message
1000 xxxx	128 to 143	\$80 to \$8F	2		Note Off
1001 xxxx	144 to 159	\$90 to \$9F	2		Note On
1010 xxxx	160 to 175	\$A0 to \$AF	2		Polyphonic Key Pressure After Touch
1011 xxxx	176 to 191	\$B0 to \$BF	2		Control Change
1011 xxxx	176 to 191	\$B0 to \$BF	2		Select Channel Mode
1100 xxxx	192 to 207	\$C0 to \$CF	1		Program Change
1101 xxxx	208 to 223	\$D0 to \$DF	1		Channel Pressure After Touch
1110 xxxx	224 to 239	\$E0 to \$EF	2		Pitch Wheel Change
1111 0000	240	\$F0	variable		System Exclusive
1111 0001	241	\$F1	-		Unimplemented
1111 0010	242	\$F2	2		Song Position Pointer
1111 0011	243	\$F3	1		Song Select
1111 0100	244	\$F4	-		Unimplemented
1111 0101	245	\$F5	-		Unimplemented
1111 0110	246	\$F6	0		Tune Request
1111 0111	247	\$F7	0		End Of Exclusive
1111 1000	248	\$F8	0		Timing Clock
1111 1001	249	\$F9	-		Unimplemented
1111 1010	250	\$FA	0		Start
1111 1011	251	\$FB	0		Continue
1111 1100	252	\$FC	0		Stop
1111 1101	253	\$FD	-		Unimplemented
1111 1110	254	\$FE	0		Active Sensing
1111 1111	255	\$FF	0		System Reset

After that unrelenting barrage of MIDI information, I feel that it is only fitting to supply you with a bit more of a detailed description. To begin, the frequency data bytes, hex values \$00 to \$7F, are obviously the note values as expressed through almost 10 complete octaves. This is one drawback of the system. 7 bits of resolution is not sufficient to allow graduations in smaller increments than semitones. One recent speculation made regarding this point stated that future revisions of the system specifications should allow note increments in graduations of cents, 1/100's of a semitone. A possible but difficult to implement idea.

Status Messages, in contrast to the note values, require more than a little explanation before becoming coherent to any degree. Therefore, there are two different Status messages that can be encountered. The Channel Message and the System Message.

The Channel Message

A Channel message is one that states a message and a specific unit number to which the message is addressed. The lower 4 bits, bits 0-3, are used for this identification. In the charts shown above, plus the descriptions to follow, this has been signified by the xxxx in place of the lower nibble in each appropriate bit pattern. This 4 bit channel designation also explains the limitation of this system to 16 channels (decimal range 0-15).

There are two types of channel messages; Voice and Mode. Voice messages control each separate units voices, which are sent over the voice channels. The Mode message defines the instruments response to the Voice messages, sent over the instruments basic channel. The Mode message can control such things as Local Control On and Off, All Notes Off, Omni Mode On or Off, Mono Mode On/Poly Mode Off, or Poly Mode On/Mono Mode Off. See message Select Channel Mode for more information.

The System Message

System messages are ones that do not pertain to any one specific channel. Being the elite of the system entitles them to speak and except all to listen, immediately. Within the realm of the System message falls the Common, Real-Time, and Exclusive messages.

Common messages are directed to all units, regardless of channel number. The Common messages are comprised of Song Position Pointer, Song Select, Tune Request, and End Of Exclusive. Each Common message consists of a single byte, of which an in depth explanation will follow promptly.

Real-Time messages are also intended to be heard by all units in the system, at any time, even during the transmission of other data by a Status byte. A Real-Time message is one in which can be tested for and ignored by the units, or acted upon

if the information is desired. Real-Time messages are comprised of the Timing Clock, Start, Continue, Stop, Active Sensing, plus System Reset. Explanations will follow.

There is only one Exclusive Message, of which carries one Status byte, plus any length of data bytes following. It is terminated by either the Common message End Of Exclusive (EOX), or any other Status byte. An Exclusive Message is one that has been incorporated to identify the manufacturer of each piece of equipment, plus allow the manufacturer to transmit whatever message they please. A form of personalized service, finally.

Status Byte Explanations

Message:**Note Off**

Pattern: 1000 xxxx

The Note Off Message, used in conjunction with the Note On message, determine when each note will start and stop for the channel affected. Following this byte are always two data bytes. The first byte determines which note within a specific octave is to be affected, as can be demonstrated in the note chart above. The second byte is the Note Off Velocity. This value is used in synthesizers capable of ADSR (attack, decay, sustain, release), to set the rate of release from the sustained level. In synthesizers not so enabled, the release is immediate.

Message:**Note On**

Pattern: 1001 xxxx

As specified above, the Note On command is used to determine which note is stuck on a specified channel. The Note On Velocity, the second data byte, has a unique use this time. It determines the loudness setting of the note value struck. In units capable of ADSR, this level of loudness will be attained within a user predetermined time limit. If the unit is incapable of ADSR, the level will be reached immediately. The maximum loudness setting, considering we are working with the lower 7 of 8 bits, is a decimal value of 127. A value of 0 would be the same as turning the Note Off. According to specs, a value of 1 is equivalent to "triple pianissimo", very quite. A value of 127 (maximum), is "triple forte", very loud. The middle of the scale is 64, which is somewhere between "mezzo-piano" and "mezzo-forte" - middle scale. If the unit is incapable of various velocity settings, a value of 64 is used and transmitted.

Message:**Polyphonic Key Pressure After Touch**

Pattern: 1010 xxxx

Polyphonic Key Pressure After Touch is really just what is implied. The new value of key pressure attained after touching it. If the value has been set before to reflect the key pressure exerted, and the user strikes the key once again, this new pressure must be reflected. This status byte has two data bytes following. The first byte is the note stuck, with the second value byte representing the pressure. The values of pressure range from 0, no pressure at all, to 127, bashed through the keyboard.

Message:Control Change

Pattern: 1011 xxxx

Control Change is a status byte that is transmitted whenever a controller mechanism is adjusted. Controller mechanisms in this context refer to foot pedals, knobs (pots), modulation wheels, sliders, and switches. Two data bytes are used after the status message to reflect this change. The first byte indicates which controller was affected, while the second byte determines the new value attained. There are four ranges allowed for various controller mechanism used, as shown below.

Value Range	Controller Type
\$00 to \$3F	Continuous (ie. Foot Pedals, Knobs, Modulation Wheels, Sliders)
\$40 to \$5F	Switches
\$60 to \$7A	Presently Undefined
\$7B to \$7F	Channel Mode Messages (see Select Channel Mode following)

The first range, Continuous, often requires a more subtle range of graduations in comparison to say a switch. Either a switch is off or on, 0 or greater than 0. Continuous devices such as a slider can graduate across the scale in as large or small increments as desired, therefore a very subtle range is required. For this reason, the Control Change message allows for three data bytes if necessary. As stated before, the first data byte determines the controller affected, with the second data byte determining the value to set. This acts as the High Byte, or increment of 256 Lower Bytes. By now you know that the third byte is the Low Byte, thereby allowing a range of 14 bits, or 16,384 graduations. The Low Byte is not mandatory, therefore it can be left off if not required.

Message:Select Channel Mode

Pattern: 1011 xxxx

Select Channel Mode is a unique message that allows alterations to the way a MIDI unit will respond to, and transmit MIDI channel messages. Two data bytes are used to allow this transformation, with the first data byte having a limited range of \$7A to \$7F. This limitation is imposed due to the fact that two different messages share the same bit pattern (see Control Change above). The second data byte is used in conjunction with the first to achieve the result desired, as shown below.

Byte 1	Byte 2	Result
\$7A	\$00	Local Control Off
\$7A	\$7F	Local Control On
\$7B	\$00	All Notes Off
\$7C	\$00	Omni Mode Off, All Notes Off
\$7D	\$00	Omni Mode On, All Notes Off
\$7E	\$0x	Mono Mode On, Poly Mode Off, All Notes Off (x = number channels)
\$7F	\$00	Poly Mode On, Mono Mode Off, All Notes Off

The term Poly refers to Polyphonic sound reproduction, which is the ability for the unit to allow more than one note to be played simultaneously. Mono refers to Monophonic sound reproduction, which is reproduction of sound that only allows one note to be played during any one time period.

Omni is a term that is used to describe the ability for a unit to respond to all system messages, or only ones addressed to it's basic channel. When Omni is Off, the unit will only listen for messages addressed to itself. When Omni is On, it will listen and act upon every message coming over the bus.

Local Control is the ability for a unit to act through, or bypass it's own circuitry for the generation of sound. With Local Control On, the synthesizer will act like it normally does. With Local Control Off, the keyboard will still produce MIDI data as it is played, but the synthesizer will not produce any sound to compliment the data. This feature was incorporated to allow keyboards to control instruments other than the synthesizer attached.

Message:Program Change

Pattern: 1100 xxxx

Program Change is used with synthesizers that have banks of memory set aside for various user chosen sounds. Often times these sounds are assigned by changing the bank or patch number currently in use. Program Change allows a reflection of a change in this bank number for the channel affected, with one data byte assigned for the bank number chosen. The bank number can have a value of 0 to 127.

Message:Channel Pressure After Touch

Pattern: 1101 xxxx

The Channel Pressure After Touch is an interesting feature that determines the average pressure values for the unit at any instant. With this average pressure determined, a desired variation of the overall timbre value or volume of the instrument can be calculated by determining the deviation from the average required. The single data byte allowed can have a value of 0, no pressure, to 127, as much pressure as possible all around.

Message:Pitch Wheel Change

Pattern: 1110 xxxx

This message is one that allows a reflection of any change in the setting of the pitch wheel. As with Continuous Controller Mechanisms such as sliders, a fairly large scale is required to reflect variations in the pitch wheel setting. For this purpose, 14 bits of resolution have been provided. One odd point to note about the MIDI system specs at this point. With the Control Change message, a resolution of 14 bits was available if required. The data bytes were read High to Low. For a Pitch Wheel Change, 14 bits are also provided, but they are read Low to High. This poorly thought out variation could lead to confusion in writing code if taken for granted. So much for conventions.

Message: System Exclusive

Pattern: 1111 0000

A System Exclusive message is one that I find to be refreshing. It allows identification of the manufacturer, and also allows the same to get his two bits in. A System Exclusive message is comprised of the first data byte signifying the identification number assigned to the manufacturer, as per the IMA (International MIDI Association In California), and as many bytes following as the manufacturer requires to tell his story. This message is terminated either by an End Of Exclusive Message (EOX – 1111 0111), or any other Status byte that happens by. Below can be found a partial list of manufacturers ID codes, as supplied by the IMA.

Manufacturer	ID
Sequential Circuits Inc.	\$01
Big Briar	\$02
Octavew/Plateau	\$03
Moog Music	\$04
Passport Designs	\$05
Lexicon	\$06
Bon Tempi	\$20
S.I.E.L.	\$21
Kawai	\$40
Roland	\$41
Korg	\$42
Yamaha	\$43

Message: Song Position Pointer

Pattern: 1111 0010

The Song Position Pointer is a 14 bit value that allows a record to be kept of the number of beats since the start of a song session. This has been incorporated for use with synthesizers equipped with a sequencer (digital recorder), or for a rhythm machine. With this feature enabled, a flag can be set to allow music to be played from a specific location within the song. The two data bytes are read Low to High.

Message: Song Select

Pattern: 1111 0011

As with the Song Position Pointer, Song Select is also meant to be used with synthesizers equipped with a sequencer, or for a rhythm machine. Song Select uses one data byte to select which song or note sequence is to be played after a Start message has been received.

Message: Tune Request

Pattern: 1111 0110

Tune Request is a throw back into the age of Analog synthesizers. This single Status byte, no data bytes, is used to request a tuning of the Analog synthesizers oscillators.

Message: End Of System Exclusive (EOX)

Pattern: 1111 0111

The EOX message is a single Status byte, no data bytes, that flags when a System Exclusive Message has been completed.

Message: Timing Clock

Pattern: 1111 1000

The Timing Clock Message is one that can be used to synchronize all sequencers and/or rhythm machines on line. The clock transmits its message at a rate of 6 messages per beat. As stated earlier, this type of message will appear at a regular intervals, regardless of the current state of other Status messages.

Message: Start

Pattern: 1111 1010

As before, this Status message is intended for use with a synthesizer equipped with a sequencer or rhythm machine. This message will inform the sequencer/rhythm machine to begin playing a pre-arranged song or note sequence from the beginning. There is only one Status Byte, no data bytes, for this message. See the Song Select Message for a little more information regarding Start.

Message: Continue

Pattern: 1111 1011

As before, the Continue message has been incorporated for use with synthesizers equipped with a sequencer or rhythm machine. This message is used to restart the current song sequence after receiving the next Timing Clock message. The sequence is then picked up from the next position in the Song Position Pointer. This message is flagged by the user pressing the Continue button on the sythesizer. As before, this message carries one Status byte and no data bytes.

Message: Stop

Pattern: 1111 1100

Again, the Stop message is for synthesizers equipped with a sequencer or rhythm machine. When received, this message tells the sequencer to stop playing its current sequence. This message carries one Status byte and no data bytes.

Message: Active Sensing

Pattern: 1111 1110

This message is one that is transmitted by any MIDI instrument on line, powered up, but not actively involved in anything. This message is transmitted once every 300 milliseconds if

there is no activity on the MIDI bus. One Status byte and no data bytes.

Message:**System Reset**

Pattern: 1111 1111

The System Reset message performs exactly as you might expect. It tells all MIDI instruments on line to perform a power up sequence to return them to a freshly powered up state.

Retrospect

The MIDI system, as has been expressed throughout this article, is a series of well thought of specifications. Through each manufacturers implementation, musicians can use instruments of different origin and expect predictable results. This is the basic flaw in the industry today. MIDI on paper appears quite explicit in its goals. But soon after implementation, the manufacturers discovered many minute points not completely taken into consideration by MIDI Version 1.0. Today, it is this problem that confronts every musician who considers the step into the world of music synthesis.

In partial explanation, manufacturers immediately found problems implementing MIDI 1.0 following its release. Although each manufacturer tried their best to work the MIDI system within the boundaries of their machines, hindsight informed us of the inevitable. The manufacturers did not work together to make sure the systems were compatible. Each operated within their own collective vacuum, producing equipment meeting untested specifications, without considering the fact that MIDI 1.0 might be vague enough to allow multiple interpretations.

When the first MIDI machines were released on the market, problems became apparent immediately. Although many of the machines were compatible, some subtle to extreme cases of incompatibility did exist. And, as with any manufacturer faced with high R&D costs, plus further misinterpretations, many of the problems went unresolved. Take for example the Yamaha DX7 music synthesizer. It's a great machine, yet its MIDI implementation is not completely compatible with say a Roland or a Korg. In this relatively new field of digital music synthesis, it is not hard to find people who have personally become victims in this rush to compatibility.

Although I may appear to hold conflicting opinions regarding the MIDI, this is not so. MIDI in general does have its fine points. But I think that it is time for the manufacturers to get together and try to work their problems out. The specifications do not have to be altered just yet, just clarified down to the finest detail. Following this, each manufacturer can regroup to produce low cost true MIDI updates for the machines currently out. The longer this move is neglected, the greater the chance of more permanent problems. So much for a bit of sage advice.

In Summation

In closing, I would like to reflect on possible extensions to the MIDI 1.0 specifications. Due to the fact that 16 and 32 bit chips have dropped in price, implementations using these chips could be considered. With these chips, high speed communications, multi tasking, and unbelievable control over vast amounts of RAM and ROM could be taken advantage of. Further to this, we have had a chance to work with the MIDI 1.0 for quite a while now. This time of reflection has enabled musicians from all over to discover most of the weak points inherent in the system. With these points in mind, plus the technology available to us today, a true implementation could be performed, with one problem. The market has already been flooded with MIDI 1.0 equipped units. A revision at this stage would make everything else obsolete, as far as current thought allows. Due to this single fact, the MIDI specifications will not be allowed revision on a radical scale for some time to come. Although it may be hard to accept this twist of fate, the human side of this story requires consideration. Musical technology is slated to stagnate for the next few years, after which time will come an age of radical change. What our dreams are composed of today is the reality of tomorrow.

Finally, I would like to thank a few people who have helped me understand MIDI more than I ever thought possible. They include Vera Barycky, who supplied me with vast amounts of hard to get information that otherwise would be inaccessible. My father, Ted Evers, for his continuous stream of information and knowledge, and my brother, John Evers, for all his related experience working within the field, from which I was able to extract some particularly critical information.

References

- | | |
|--|--------------------------|
| The MIDI Handbook | by Paul Vytas |
| MIDI Specification 1.0 | by the IMA in California |
| The Roland Juno-60 Manual | by Roland |
| The Roland JX-3P Implementation | by Roland |
| The Yamaha RX15 Digital Rhythm Programmer Manual | by Yamaha |
| Keyboard Magazine | June 84 & July 85 |
| Mix Magazine | August 1984 |
| Computing Now | December 1984 |
| A stack of IMA newsletters | by the IMA in California |

Real-World Interfacing With The REL64 Cartridge

James E. LaPorte
Blacksburg, VA

Dr. LaPorte is Assistant Professor of Industrial Arts at The Virginia Polytechnic Institute and State University. M.Ed.

Sooner or later, there becomes an overwhelming desire among most computer enthusiasts to connect their machine to the outside world and control lights, motors, and other devices. The recent increase in articles devoted to this subject in virtually all computer publications is evidence of such a desire. On the other hand, many of these articles require at least some knowledge of electronic components, the ability to prepare printed circuit boards, and perhaps even a moderate knowledge of electronic theory. In addition, it takes a bit of bravery to connect the prescribed hardware to the computer, anticipating ruining the system due to an error.

There is a product that has been available for over two years that reduces some of the above problems. Originally called the VIC-REL, it is now called the REL 64 Relay Cartridge and is marketed by Handic Software, Inc. (400 Paterson Plank Rd., Carlstadt, NJ 07072 - \$39.95 U.S.). The cartridge plugs into the user port and is compatible with any Commodore computer that has one.

Details of the Cartridge

As illustrated in Figure 1, there are a series of terminal screws on the end of the cartridge to which wires can be attached. The first six pairs of terminals (nos. 1-12) are equivalent to the terminals on six simple switches. Thus, six separate output devices can be controlled.

In addition, the cartridge is capable of detecting input from two separate sources. This is accomplished by sensing the presence or absence of 5 volts d. c. across terminal pairs 17-18 or 19-20. Though care must be taken to avoid short circuits, this voltage can be obtained from the power supply of the computer across terminals 14 and 15. Terminals 13 and 16 are unused.

Programming for Output

Only two memory locations need to be accessed to use the REL 64 cartridge. Though the specific locations for the Commodore 64 will be given as examples, the equivalent locations for the VIC-20, Plus 4, and other models with a user port can be readily found in programming reference guides.

The first of these locations is the data direction register (DDR) for the user port, 56579 decimal. When a bit at the user port is to be used for output, the operating system in the computer requires that the corresponding bit in the DDR be set high (on). Likewise, if a bit is to be used for input, the corresponding bit in the DDR is set low (off). The cartridge uses the first six bits to control the switch terminals, thus they are outputs from the computer. The remaining two bits are input. So, to configure the user port in a manner consistent with the cartridge, the number 63 is POKed into the DDR, turning the first six bits on.

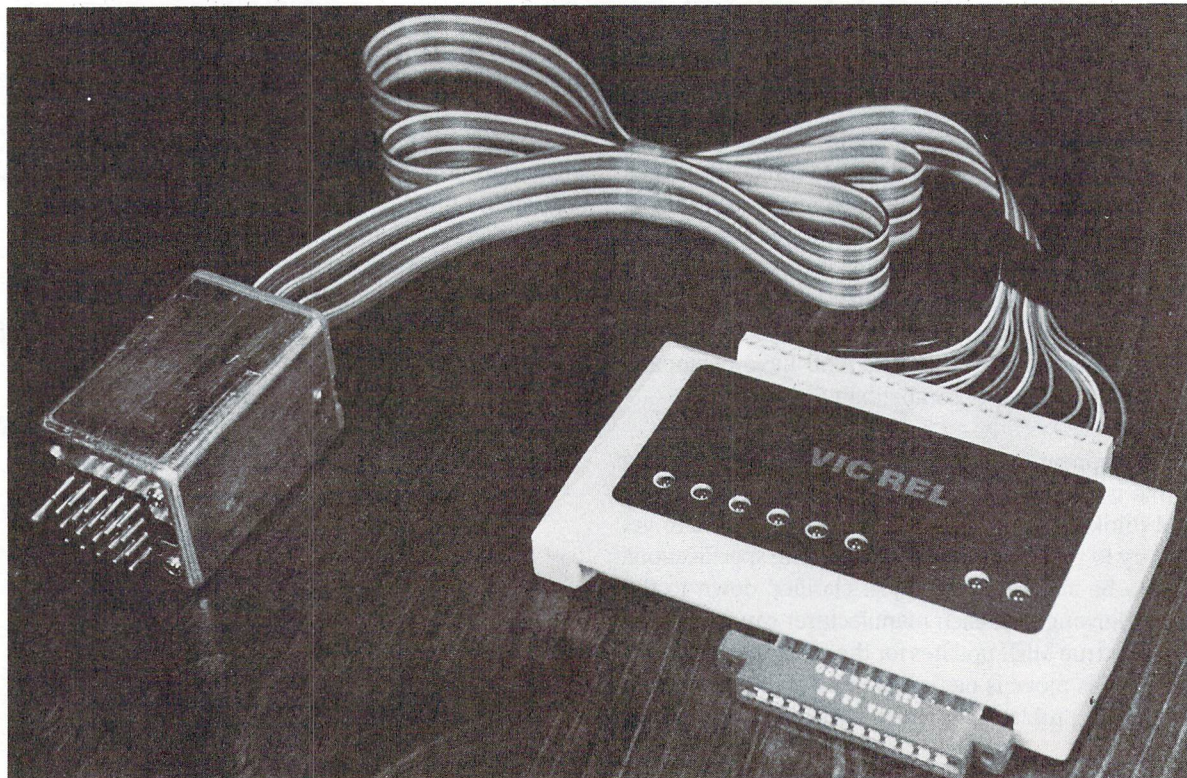


Figure 1. The VIC-REL (REL 64) Cartridge with Ribbon Cable and Discarded Telephone Connector Attached.

Once the port has been properly configured, control of output is accomplished by setting the appropriate bit at the user port (data port B) high. On the C64, this is decimal 56577. For example, if a 1 is POKed into this location, the first bit is turned on and there will be electrical continuity between terminal screws 1 and 2 of the cartridge. Thus, a light bulb or similar device with one side of the supply in series with these terminals (like a switch) would be turned on. Likewise, a 2 POKed into 56577 would turn on the second bit and provide electrical continuity between terminals 3 and 4, and so forth.

Programming for Input

The last two bits (6 and 7) of the user port are assigned for input. If a voltage between 5 and 12 volts d. c. is applied across terminals 17 and 18, then bit 6 is changed to a low state (off). If the same voltage is applied across terminals 19 and 20, then bit seven is set low. Note that this is opposite to what would normally be expected – the bit is off when voltage is applied.

To read input, the respective bit must be exclusively PEEKed so that the changing status of the other bits does not interfere. This is done by ANDing the bit during the PEEK. For example, to read the status of bit 6, the following program line would be used:

```
10 J = PEEK (56577) AND 64
```

If a zero is returned by the variable J, then the bit is off and 5 volts is present across terminals 17 and 18. If J returns a 64, then no voltage is present. Likewise, for bit 7, this line would be used:

```
20 K = PEEK (56577) AND 128
```

Variable K will return a zero if the bit is on and 128 if the bit is off.

Controlling Heavy Loads

The output capability of the 64 Relay Cartridge is designed for a maximum of 24 volts at 10 watts. Depending on the specifics of the intended application, this may be a serious limitation. There is, however, a relatively easy method to solve this problem by using a relay.

A relay is an electromagnetic switch. When current flows in the coil of the relay, it draws a contact toward it, completing an electrical circuit. Since the contacts of the relay are completely isolated from the coil, virtually any load desired can be controlled. The only requirement is that the voltage and power necessary for the coil of the relay does not exceed the maximum limits of 24 volts at 10 watts.

Radio Shack and other electronic component suppliers have a wide variety of relays that are suitable. The Radio Shack #275-218 relay, for example, has contacts that will handle a device rated at 10 amps. at 125 volts. This means the relay can control a load of up to 1250 watts (Wattage = Voltage x Amperes). On the other hand, this particular relay has a coil designed to operate on 12 volts at .075 amps. (0.9 watts) – well within the maximum limits of the cartridge.

The only problem in implementing relays is that a separate power supply for the relay coils must be used. Since the 12 volts necessary for the relay coils is quite common, a suitable supply might be found in a discarded "power pack" from a portable appliance, radio, etc. Such a supply might also be purchased from surplus electronic dealers. Of course, the power supply can be easily built with a minimum of knowledge.

Regardless of how the supply is secured, it must have sufficient power to supply the number of relays to be used. If six relays of the type referenced above are used, thus utilizing the full capability of the REL 64 Cartridge, the supply must deliver 450 milliamperes (6 times .075 amps = .450 amps).

A schematic for one relay is shown in Figure 2. As mentioned previously, up to six relays could be connected. The power supply connections would simply be extended to the additional relays and additional pairs of terminals (3&4, 5&6, etc.) on the cartridge would be connected.

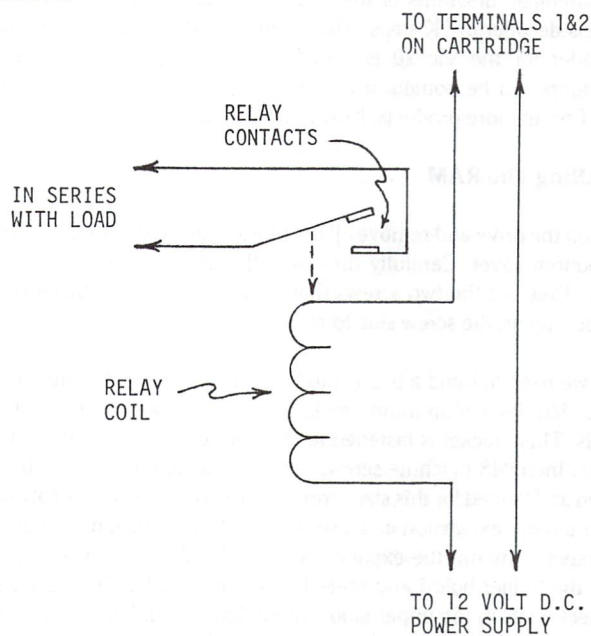


Figure 2: Schematic of connections for a relay to control heavy loads

Conclusion

The REL 64 cartridge is an easy way for a computer enthusiast to get involved in controlling real world devices with minimal knowledge of electricity. Burglar alarm systems, weather stations, model railroads, and automatic door openers are but a few of the many possible applications.

The cartridge is designed in such a way that the electrical connections are isolated from the computer. Though a gross wiring error could damage the cartridge, it cannot damage the computer. The only exception to this is the 5 volts that are available across terminals 14 and 15. Though having this voltage available makes it easy to utilize the input functions of the cartridge, care must be taken since a short across these two terminals could damage the computer.

Though the cartridge is fixed to control six outputs and sense two inputs, this should not be a major limiting factor, especially for the beginner. In addition, relays allow the development of circuits with virtually any current or voltage desired, reducing the limitation of rather low current and voltage maximums of the cartridge itself.

Several of the cartridges have been used in teaching college students the basic concepts of computer interfacing in the Industrial Arts Program Area at Virginia Tech. Even students with limited experience with microcomputers have been able to construct interesting and challenging computer-controlled projects. What is more, they seem to leave the course with a clear understanding of bits and bytes, unlike that when traditional methods are used.

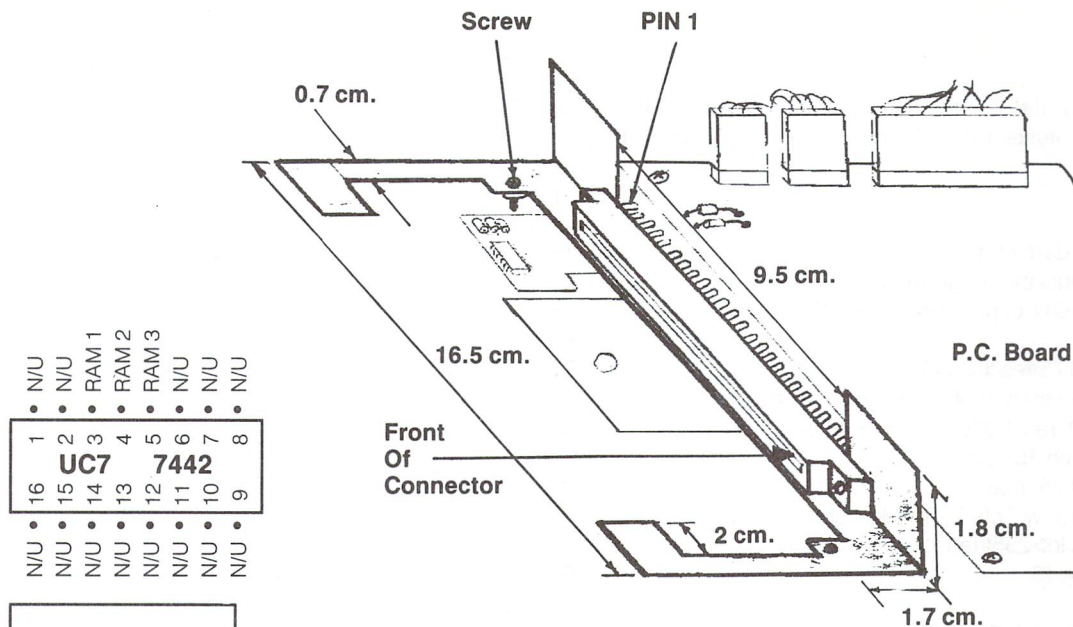


Figure 1: Bracket mounted on top front of 1541

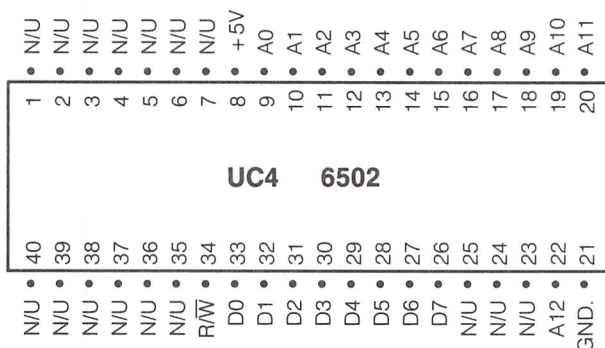


Figure 2: Showing UNDERSIDE of
1541 Main Board (partial view)

Listing 1

BASIC program which uses the program in 1541 RAM (Listing 2) to create an extra header block on any track. Lines 30-50 load the program into drive memory using the "DOS Executor" feature. Delete these lines if using the loader in Listing 2 to put the program into drive memory.

```

AD 10 rem*****this program creates an extra
    header block on any track*****
DG 20 rem*****it does not create a data
    block for the extra header block*****
MN 30 open15,8,15
CL 40 print#15, "&:extra header"
PP 50 for x = 0 to 4000: next: close15
IM 60 print "please insert disk to have extra sector
    header block added and press";
KG 70 print " return."
IB 80 get a$: if a$ = " " then 80
DB 90 if a$ <> chr$(13) then 80
  
```

```

IA 100 print "do you want the disk formatted;yes or no"
BM 110 get a$: if a$ = " " then 110
CK 120 a$ = left$(a$,1)
DE 130 if a$ <> "y" and a$ <> "n" then 100
BN 140 if a$ = "n" then 190
JJ 150 input "please print disk name "; a$
DC 160 input "please print two digit i.d. code "; b$
IG 170 open15,8,15
HL 180 print#15, "n: "; a$; ", "; b$: close15
JP 190 open15,8,15: print#15, "i ": gosub 400: close15
IL 200 input "what track do you want the extra
    header block on, 1-35 "; c
FI 210 if c < 1 or c > 35 then 200
BP 220 if c > 0 and c < 18 then d = 21
CL 230 if c > 17 and c < 25 then d = 19
JK 240 if c > 24 and c < 31 then d = 18
HL 250 if c > 30 and c < 36 then d = 17
CM 260 open15,8,15
CD 270 print#15, "m-w" chr$(254)chr$(19)chr$(02)
    chr$(c)chr$(d)
OL 280 gosub 400
ON 290 e = d - 1: count = 0
JE 300 print#15, "m-w" chr$(00)chr$(04)chr$(03)chr$(76)
    chr$(03)chr$(08)
MN 310 gosub 400
LH 320 print#15, "m-w" chr$(08)chr$(00)chr$(02)
    chr$(c)chr$(e)
AP 330 gosub 400
CO 340 print#15, "m-w" chr$(01)chr$(00)chr$(01)chr$(224)
EA 350 gosub 400
HM 360 if f$ = "00" then print "done,all ok ": close15: end
NC 370 if count = 10 then 390
OM 380 count = count + 1: go to 300
NJ 390 gosub 400: goto 420
NO 400 input#15, f$, g$, h$, i$
CH 410 if f$ = "00" then return
DJ 420 print f$, g$, h$, i$, " failed"
PI 430 close15: end
  
```


Listing 2.

The BASIC loader for the "header block" machine code. This program will put the program directly into the new drive RAM at \$0800.

```
IK 10 rem* data loader for "header block"
JH 11 rem* this puts the program directly
PL 12 rem* into 1541 expansion ram at 0800
BI 13 :
HK 20 for i = 1 to 175:read a:c = c + a:next
MN 30 if c<>20663 then print "!data error!":end
BH 40 hi=8:lo=0: rem 0800 in drive memory
AK 50 restore: open 15,8,15
LA 60 for i = 1 to 175: read a
IO 70 print#15, "m-w: " chr$(lo)chr$(hi)chr$(1)chr$(a)
LM 80 lo = lo + 1:if lo>255then lo = 0:hi = hi + 1
CM 90 next i: close 15
IN 100 :
FL 1000 data 76, 160, 234, 173, 254, 19, 133, 24
ID 1010 data 173, 255, 19, 133, 25, 169, 0, 69
OK 1020 data 22, 69, 23, 69, 24, 69, 25, 133
BD 1030 data 26, 32, 52, 249, 162, 0, 181, 36
EA 1040 data 157, 0, 16, 232, 224, 10, 208, 246
FG 1050 data 173, 254, 19, 133, 6, 174, 255, 19
NA 1060 data 202, 134, 7, 169, 0, 133, 61, 133
JF 1070 data 51, 133, 48, 169, 6, 133, 50, 169
OM 1080 data 3, 133, 49, 32, 16, 245, 32, 86
JK 1090 data 245, 80, 254, 184, 173, 1, 28, 145
AM 1100 data 48, 200, 208, 245, 160, 186, 80, 254
LM 1110 data 184, 173, 1, 28, 153, 0, 1, 200
HD 1120 data 208, 244, 184, 80, 254, 184, 173, 1
MH 1130 data 28, 201, 85, 240, 246, 173, 12, 28
IL 1140 data 41, 31, 9, 192, 141, 12, 28, 169
DF 1150 data 255, 141, 3, 28, 169, 255, 141, 1
BO 1160 data 28, 162, 5, 80, 254, 184, 202, 208
OF 1170 data 250, 162, 0, 80, 254, 184, 189, 0
OI 1180 data 16, 141, 1, 28, 232, 224, 10, 208
ON 1190 data 242, 162, 9, 80, 254, 184, 169, 85
HK 1200 data 141, 1, 28, 202, 208, 245, 32, 0
BA 1210 data 254, 169, 1, 76, 105, 249, 69
```

Listing 3.

The BASIC code for the C64 which will format a disk with 38 tracks. It uses the program in Listing 4 which resides in the new drive RAM. Lines 10-20 assume a "DOS Executor" file on disk; delete them if using the loader in Listing 4.

```
IM 10 open15,8,15
HA 20 print#15, "&:format 38"
CM 30 print "please insert disk to have extra tracks ";
CE 40 print "and press return."
HP 50 get a$:if a$ = "" then 50
CP 60 if a$<>chr$(13) then 50
JE 70 input "please print disk name ";a$
DN 80 input "please print two digit i.d. code ";b$
PA 90 print#15, "n: ";a$," ";b$:gosub260
```

```
HP 100 count = 0
EB 110 rem ;put jmp $0803 in buffer #3($0600)
LJ 120 print#15, "m-w " chr$(00)chr$(06)chr$(03)
    chr$(76)chr$(03)chr$(08)
KD 130 gosub 260
HE 140 rem ;put track 35 and sector 0 in $000c/d of
    the header table
IB 150 print#15, "m-w " chr$(12)chr$(00)chr$(02)
    chr$(35)chr$(0)
IF 160 gosub 260
EA 170 rem ;put execute code in $0003
GE 180 print#15, "m-w " chr$(03)chr$(00)chr$(01)chr$(224)
MH 190 rem ;now read the disk controller error code
ON 200 print#15, "m-r " chr$(03)chr$(00)
DH 210 get#15,er$:er = asc(er$)
PO 220 if er = 01 then print "done,all ok ":close15:end
IJ 230 if count = 10 then 300
EE 240 count = count + 1:go to 120
BC 250 gosub 260:goto 280
BG 260 input#15,f$,g$,h$,i$
GO 270 if f$ = "00" then return
NJ 280 print f$:g$:h$:i$; " failed"
DA 290 close15:end
CK 300 if er < 17 then print "controller error # ";er
CO 310 goto 250
```

Listing 4

The loader for "format38", which puts the new formatting routine into drive memory.

```
ME 10 rem* data loader for "format38"
JH 11 rem* this puts the program directly
PL 12 rem* into 1541 expansion ram at 0800
BI 13 :
OK 20 for i = 1 to 752:read a:c = c + a:next
KO 30 if c<>83435 then print "!data error!":end
FC 35 print "data ok, now loading to drive ram"
BH 40 hi=8:lo=0: rem 0800 in drive memory
AK 50 restore: open 15,8,15
CB 60 for i = 1 to 752: read a
IO 70 print#15, "m-w: " chr$(lo)chr$(hi)chr$(1)chr$(a)
LM 80 lo = lo + 1:if lo>255then lo = 0:hi = hi + 1
CM 90 next i: close 15
IN 100 :
AK 1000 data 76, 160, 234, 165, 34, 133, 81, 169
JC 1010 data 20, 141, 32, 6, 169, 64, 141, 33
II 1020 data 6, 169, 15, 141, 34, 6, 169, 16
GN 1030 data 133, 67, 32, 207, 10, 32, 207, 10
LI 1040 data 230, 81, 165, 81, 201, 36, 144, 242
BM 1050 data 32, 163, 253, 32, 195, 253, 169, 85
LL 1060 data 141, 1, 28, 32, 195, 253, 32, 0
CC 1070 data 254, 32, 86, 245, 169, 64, 13, 11
DH 1080 data 24, 141, 11, 24, 169, 98, 141, 6
JH 1090 data 24, 169, 0, 141, 7, 24, 141, 5
ID 1100 data 24, 160, 0, 162, 0, 44, 0, 28
AK 1110 data 48, 251, 44, 0, 28, 16, 251, 173
CF 1120 data 4, 24, 44, 0, 28, 16, 17, 173
CH 1130 data 13, 24, 10, 16, 245, 232, 208, 239
```


NH	1140 data 200, 208, 236, 169, 2, 76, 189, 10
PF	1150 data 134, 113, 132, 114, 162, 0, 160, 0
DI	1160 data 173, 4, 24, 44, 0, 28, 48, 17
BK	1170 data 173, 13, 24, 10, 16, 245, 232, 208
AC	1180 data 239, 200, 208, 236, 169, 2, 76, 189
LI	1190 data 10, 56, 138, 229, 113, 170, 133, 112
MM	1200 data 152, 229, 114, 168, 133, 113, 16, 11
GC	1210 data 73, 255, 168, 138, 73, 255, 170, 232
CK	1220 data 208, 1, 200, 152, 208, 4, 224, 4
KM	1230 data 144, 24, 6, 112, 38, 113, 24, 165
HO	1240 data 112, 109, 33, 6, 141, 33, 6, 165
OF	1250 data 113, 109, 34, 6, 141, 34, 6, 76
JF	1260 data 40, 8, 162, 0, 160, 0, 184, 173
DA	1270 data 0, 28, 16, 14, 80, 249, 184, 232
LH	1280 data 208, 245, 200, 208, 242, 169, 3, 76
KA	1290 data 189, 10, 138, 10, 141, 37, 6, 152
LG	1300 data 42, 141, 36, 6, 169, 191, 45, 11
FB	1310 data 24, 141, 11, 24, 169, 102, 141, 38
CE	1320 data 6, 166, 67, 160, 0, 152, 24, 109
NB	1330 data 38, 6, 144, 1, 200, 200, 202, 208
BG	1340 data 245, 73, 255, 56, 105, 0, 24, 109
NJ	1350 data 37, 6, 176, 3, 206, 36, 6, 170
DH	1360 data 152, 73, 255, 56, 105, 0, 24, 109
JK	1370 data 36, 6, 16, 5, 169, 4, 76, 189
LK	1380 data 10, 168, 138, 162, 0, 56, 229, 67
II	1390 data 176, 3, 136, 48, 3, 232, 208, 245
HE	1400 data 142, 38, 6, 224, 4, 176, 5, 169
NJ	1410 data 5, 76, 189, 10, 24, 101, 67, 141
GM	1420 data 39, 6, 169, 0, 141, 40, 6, 160
HJ	1430 data 0, 166, 61, 165, 57, 153, 0, 3
JA	1440 data 200, 200, 173, 40, 6, 153, 0, 3
BK	1450 data 200, 165, 81, 153, 0, 3, 200, 181
KP	1460 data 19, 153, 0, 3, 200, 181, 18, 153
FK	1470 data 0, 3, 200, 169, 15, 153, 0, 3
NC	1480 data 200, 153, 0, 3, 200, 169, 0, 89
EP	1490 data 250, 2, 89, 251, 2, 89, 252, 2
HE	1500 data 89, 253, 2, 153, 249, 2, 238, 40
DE	1510 data 6, 173, 40, 6, 197, 67, 144, 187
KC	1520 data 152, 72, 232, 138, 157, 0, 5, 232
JP	1530 data 208, 250, 169, 3, 133, 49, 32, 48
FF	1540 data 254, 104, 168, 136, 32, 229, 253, 32
MD	1550 data 245, 253, 169, 5, 133, 49, 32, 233
LE	1560 data 245, 133, 58, 32, 143, 247, 169, 0
EI	1570 data 133, 50, 32, 14, 254, 169, 255, 141
KC	1580 data 1, 28, 162, 5, 80, 254, 184, 202
DL	1590 data 208, 250, 162, 10, 164, 50, 80, 254
NL	1600 data 184, 185, 0, 3, 141, 1, 28, 200
AM	1610 data 202, 208, 243, 162, 9, 80, 254, 184
LI	1620 data 169, 85, 141, 1, 28, 202, 208, 245
CO	1630 data 169, 255, 162, 5, 80, 254, 184, 141
BH	1640 data 1, 28, 202, 208, 247, 162, 187, 80
LA	1650 data 254, 184, 189, 0, 1, 141, 1, 28
CO	1660 data 232, 208, 244, 160, 0, 80, 254, 184
NL	1670 data 177, 48, 141, 1, 28, 200, 208, 245
GP	1680 data 169, 85, 174, 38, 6, 80, 254, 184
LJ	1690 data 141, 1, 28, 202, 208, 247, 165, 50
LK	1700 data 24, 105, 10, 133, 50, 206, 40, 6
KP	1710 data 208, 147, 80, 254, 184, 80, 254, 184
NL	1720 data 32, 0, 254, 169, 200, 141, 35, 6
KP	1730 data 169, 0, 133, 48, 169, 3, 133, 49
AC	1740 data 165, 67, 141, 40, 6, 32, 86, 245

DC	1750 data 162, 10, 160, 0, 80, 254, 184, 173
OK	1760 data 1, 28, 209, 48, 208, 14, 200, 202
OG	1770 data 208, 242, 24, 165, 48, 105, 10, 133
LC	1780 data 48, 76, 126, 10, 206, 35, 6, 208
LD	1790 data 207, 169, 6, 76, 189, 10, 32, 86
AP	1800 data 245, 160, 187, 80, 254, 184, 173, 1
KC	1810 data 28, 217, 0, 1, 208, 230, 200, 208
FP	1820 data 242, 162, 252, 80, 254, 184, 173, 1
AE	1830 data 28, 217, 0, 5, 208, 214, 200, 202
OJ	1840 data 208, 241, 206, 40, 6, 208, 174, 165
NI	1850 data 81, 201, 38, 176, 3, 76, 26, 8
IK	1860 data 169, 255, 133, 81, 169, 0, 133, 80
OG	1870 data 169, 1, 76, 105, 249, 206, 32, 6
BG	1880 data 240, 3, 76, 40, 8, 160, 255, 132
NN	1890 data 81, 200, 132, 80, 76, 105, 249, 174
PF	1900 data 0, 28, 232, 138, 41, 3, 141, 255
DM	1910 data 19, 173, 0, 28, 41, 252, 13, 255
DL	1920 data 19, 141, 0, 28, 160, 5, 162, 255
LL	1930 data 202, 208, 253, 136, 208, 250, 96, 32

Listing 5

Changes to make to the "DOS Executor" program (July 85 Transactor) to allow it to work with the expansion RAM.

```

change line 370 print "bam buffer-1792 $0700"
add line 375 print "expansion ram-2048 $0800" d$
change line 400 ifval(dm$)>7419 then print d$d$ "not higher then
7419": print u$u$u$u$: goto 380
add line 461 ifcnt=0 then cnt=1: goto 470
add line 462 dm=dm+250
add line 463 hi=int(dm/256): lo=dm-hi*256
add line 501 ifcnt=0 then cnt=1: goto 510
add line 502 dm=dm+250
add line 503 hi=int(dm/256): lo=dm-hi*256

HJ 100 rem open1,8,1,"0:format38.obj
EB 110 rem save"@0:format38.pal",8
IP 120 sys700 ;pal 64 assembler
ML 130 blocks free.
OJ 140 this program lives inside the 1541
GK 150 and will format with 38 tracks
CL 160 it should be used with its basic
KO 170 counterpart in the c64
KC 180 ;
GC 190 .opt oo
GL 200 * = $0800
IE 210 ;
DL 220 jmp $eaa0 ;cold start of disk drive
MJ 230 lda $22 ;current track #
NE 240 sta $51 ;track # for formatting
PK 250 lda #$14 ;set to allow 20 errors
IM 260 sta $0620
KG 270 lda #$40 ;set $0621/2 to 4000 as a
BN 280 sta $0621 ;guess to the # of bytes
on half a track

NJ 290 lda #$0f
CP 300 sta $0622
GM 310 lda #$10 ;set to allow 16 sectors

```


on these tracks				GK	910 now	sec	;caculate time
LA	320	sta	\$43	ME	920	txa	
KL	330 bigger	jsr	step	PD	930	sbc	\$71
FD	340	jsr	step	HH	940	tax	
FB	350	inc	\$51	OH	950	sta	\$70
JM	360	lda	\$51	HH	960	tya	
			;if not on track 36 then move head	KG	970	sbc	\$72
AP	370	cmp	#\$24	DK	980	tay	
DI	380	bcc	bigger	JK	990	sta	\$71
MP	390 ;			KG	1000	bpl	pos
FD	400 start	jsr	\$fda3				;is the difference postive
KF	410	jsr	\$fdc3	KJ	1010	eor	#\$ff
KO	420	lda	#\$55	LM	1020	tay	
EJ	430	sta	\$1c01	KL	1030	txa	
NO	440	jsr	\$fdc3	IL	1040	eor	#\$ff
ME	450	jsr	\$fe00	FO	1050	tax	
FC	460	jsr	\$f556	AA	1060	inx	
ID	470	lda	#\$40	JG	1070	bne	pos
MM	480	ora	\$180b	IB	1080	iny	
AM	490	sta	\$180b	FF	1090 pos	tya	
BD	500	lda	#\$62	BL	1100	bne	tryagain
IM	510	sta	\$1806	BH	1110	cpx	#4
DP	520	lda	#0				;is the difference greater then 4
NN	530	sta	\$1807	OJ	1120	bcc	ok
FO	540	sta	\$1805	ND	1130 tryagain	asl	\$70
ED	550	ldy	#0	KF	1140	rol	\$71
HH	560	ldx	#0	EP	1150	clc	
JG	570 sync	bit	\$1c00	CB	1160	lda	\$70
OM	580	bmi	sync	LA	1170	adc	\$0621
MG	590 nonsync	bit	\$1c00	BG	1180	sta	\$0621
NJ	600	bpl	nonsync	DD	1190	lda	\$71
GF	610 reset	lda	\$1804	KC	1200	adc	\$0622
HH	620 read	bit	\$1c00	AI	1210	sta	\$0622
GP	630	bpl	here	JN	1220	jmp	start
KB	640	lda	\$180d	EE	1230 ;		
GH	650	asl	a	AJ	1240 ok	ldx	#0
PL	660	bpl	read	NC	1250	ldy	#0
KH	670	inx		OK	1260	clv	
HG	680	bne	reset	CM	1270 test	lda	\$1c00
CJ	690	iny		HN	1280	bpl	timer
LH	700	bne	reset	DM	1290	bvc	test
FL	710	lda	#2	GN	1300	clv	
KC	720	jmp	error	KP	1310	inx	
AF	730 ;			BN	1320	bne	test
EP	740 here	stx	\$71	CB	1330	iny	
MB	750	sty	\$72	FO	1340	bne	test
LB	760	ldx	#0	HD	1350	lda	#3
NE	770	ldy	#0	KK	1360	jmp	error
JI	780 reset1	lda	\$1804	AN	1370 ;		
CL	790 read1	bit	\$1c00	KJ	1380 timer	txa	
AK	800	bmi	now	KF	1390	asl	a
EM	810	lda	\$180d	BE	1400	sta	\$0625
AC	820	asl	a	JD	1410	tya	
LM	830	bpl	read1	OI	1420	rol	a
EC	840	inx		OF	1430	sta	\$0624
DH	850	bne	reset1	PI	1440	lda	#\$bf
MD	860	iny		CP	1450	and	\$180b
HI	870	bne	reset1	OD	1460	sta	\$180b
PF	880	lda	#2				;bytes on this track ;by the number of sectors
EN	890	jmp	error	LA	1470	lda	#\$66
KP	900 ;			CJ	1480	sta	\$0626

LL	1490	ldx	\$43		
HC	1500	ldy	#0		
NJ	1510	tya			
MO	1520	cac1	clc		
IH	1530	adc	\$0626		
DL	1540	bcc	cac2		
OO	1550	iny			
PH	1560	cac2	iny		
JN	1570	dex			
AA	1580	bne	cac1		
ON	1590	eor	#\$ff		
BM	1600	sec			
HC	1610	adc	#0		
KM	1620	clc			
LN	1630	adc	\$0625		
LF	1640	bcs	cac3		
HP	1650	dec	\$0624		
PM	1660	cac3	tax		
ND	1670	tya			
ID	1680	eor	#\$ff		
LB	1690	sec			
BI	1700	adc	#0		
EC	1710	clc			
ED	1720	adc	\$0624		
EM	1730	bpl	cac4		
PL	1740	lda	#4		
AD	1750	jmp	error		
GF	1760				
CE	1770	cac4	tay		
IK	1780	txa			
FE	1790	ldx	#0		
DB	1800	cac5	sec		
PK	1810	sbc	\$43		
LB	1820	bcs	cac6		
BO	1830	dey			
JC	1840	bmi	cac7		
BK	1850	cac6	inx		
IC	1860	bne	cac5		
AA	1870	cac7	stx	\$0626	
JL	1880	cpx	#4		
CI	1890	bcs	headers		
BG	1900	lda	#5		
AN	1910	jmp	error		
GP	1920				
MN	1930	headers	clc		
DC	1940	adc	\$43		
FG	1950	sta	\$0627		
DJ	1960	lda	#0		
OH	1970	sta	\$0628		
HA	1980	ldy	#0		
AO	1990	ldx	\$3d		
PH	2000	header1	lda	\$39	
AP	2010	sta	\$0300,y		
PN	2020	iny			
OM	2030	iny			
NN	2040	lda	\$0628		
IB	2050	sta	\$0300,y		
MO	2060	iny			
OC	2070	lda	\$51		
GD	2080	sta	\$0300,y		
KA	2090	iny			
EC	2100	lda	\$13,x		
EF	2110	sta	\$0300,y		
IC	2120	iny			
PD	2130	lda	\$12,x		
CH	2140	sta	\$0300,y		
GE	2150	iny			
CL	2160	lda	#\$0f		
AJ	2170	sta	\$0300,y		
EG	2180	iny			
EK	2190	sta	\$0300,y		
IH	2200	iny			
PK	2210	lda	#0		
OB	2220	eor	\$02fa,y		
MC	2230	eor	\$02fb,y		
KD	2240	eor	\$02fc,y		
IE	2250	eor	\$02fd,y		
LK	2260	sta	\$02f9,y		
CG	2270	inc	\$0628		
GH	2280	lda	\$0628		
EM	2290	cmp	\$43		
GH	2300	bcc	header1		
NL	2310	tya			
MI	2320	pha			
GP	2330	inx			
IN	2340	txa			
NG	2350	data	sta	\$0500,x	
OG	2360	inx			
HE	2370	bne	data		
ND	2380	lda	#3		
JE	2390	sta	\$31		
FO	2400	jsr	\$fe30		
CP	2410	pla			
DE	2420	tay			
JD	2430	dey			
CN	2440	jsr	\$fde5		
CP	2450	jsr	\$fdf5		
BP	2460	lda	#5		
JG	2470	sta	\$31		
CE	2480	jsr	\$f5e9		
OJ	2490	sta	\$3a		
IO	2500	jsr	\$f78f		
JL	2510	lda	#0		
OJ	2520	sta	\$32		
IJ	2530	jsr	\$fe0e		
AO	2540	write	lda	#\$ff	
MN	2550	sta	\$1c01		
KF	2560	ldx	#5		
PB	2570	wait1	bvc	wait1	
GN	2580	clv			
FN	2590	dex			
OJ	2600	bne	wait1		
PI	2610	ldx	#\$0a		
IC	2620	ldy	\$32		
CG	2630	wait2	bvc	wait2	

PK	2640	clv		;write out header	BJ	3220	lda	\$43		;num of sectors on this track
CD	2650	lda	\$0300,y		KG	3230	sta	\$0628		
KE	2660	sta	\$1c01		NI	3240 cont	jsr	\$f556		;wait for sync
OE	2670	iny			FH	3250	ldx	#\$0a		
PC	2680	dex			HA	3260	ldy	#0		
KP	2690	bne	wait2		LI	3270 chk1	bvc	chk1		
HE	2700	ldx	#9	;header gap size	CJ	3280	clv			
JL	2710 wait3	bvc	wait3		CI	3290	lda	\$1c01		
CG	2720	clv			CP	3300	cmp	(\$30),y		;check header
PE	2730	lda	#\$55	;header gap	AE	3310	bne	error1		
KJ	2740	sta	\$1c01		IN	3320	iny			
FH	2750	dex			JL	3330	dex			
CE	2760	bne	wait3		GA	3340	bne	chk1		
GO	2770	lda	#\$ff	;sync mark	MI	3350	clc			
AA	2780	ldx	#5	;sync size	CK	3360	lda	\$30		
AB	2790 wait4	bvc	wait4		DI	3370	adc	#\$0a		
CL	2800	clv			EP	3380	sta	\$30		
AO	2810	sta	\$1c01		FH	3390	jmp	more		
LL	2820	dex			OL	3400 ;				
KI	2830	bne	wait4		HJ	3410 error1	dec	\$0623		
DI	2840	ldx	#\$bb	;overflow buffer – write data block	BO	3420	bne	try		
DF	2850 wait5	bvc	wait5		NF	3430	lda	#6		
OO	2860	clv			KM	3440	jmp	error		
FA	2870	lda	\$0100,x		AP	3450 ;				
GC	2880	sta	\$1c01		LM	3460 more	jsr	\$f556		;wait for sync
GC	2890	inx			GL	3470	ldy	#\$bb		;overflow buffer—we are checking
CN	2900	bne	wait5							;the data block
JK	2910	ldy	#0		BD	3480 chk2	bvc	chk2		
AK	2920 wait6	bvc	wait6		EG	3490	clv			
ED	2930	clv			EF	3500	lda	\$1c01		
FB	2940	lda	(\$30),y	;buffer #2	NM	3510	cmp	\$0100,y		
MG	2950	sta	\$1c01		CB	3520	bne	error1		
AH	2960	iny			KK	3530	iny			
KB	2970	bne	wait6		CN	3540	bne	chk2		
BJ	2980	lda	#\$55	;tail gap	LO	3550	ldx	#\$fc		
BO	2990	ldx	\$0626	;number of tail gap bytes	HL	3560 chk3	bvc	chk3		
HP	3000 wait7	bvc	wait7		EL	3570	clv			
EI	3010	clv			EK	3580	lda	\$1c01		
CL	3020	sta	\$1c01		II	3590	cmp	\$0500,y		;buffer #2 data
NI	3030	dex			CG	3600	bne	error1		
CG	3040	bne	wait7		KP	3610	iny			
CH	3050	lda	\$32		LN	3620	dex			
KG	3060	clc			AD	3630	bne	chk3		
HF	3070	adc	#\$0a		NO	3640	dec	\$0628		;sector counter
OM	3080	sta	\$32		OF	3650	bne	cont		
LJ	3090	dec	\$0628		EG	3660	lda	\$51		;track number
JA	3100	bne	write		DL	3670	cmp	#\$26		;are 38 tracks done
MG	3110 wait8	bvc	wait8		OC	3680	bcs	done		
CP	3120	clv			JF	3690	jmp	bigger		;do more tracks
HI	3130 wait9	bvc	wait9		KO	3700 ;				
GA	3140	clv			LI	3710 done	lda	#\$ff		;reset values
IN	3150	jsr	\$fe00	;kill write mode	PE	3720	sta	\$51		
MO	3160	lda	#\$c8	;now verify track with 200 trys	NH	3730	lda	#0		
JC	3170	sta	\$0623		AG	3740	sta	\$50		
JA	3180 try	lda	#0	;set buffer pointer	DB	3750	lda	#1		;job done ok
GD	3190	sta	\$30		GE	3760	jmp	\$f969		;main error handler
BH	3200	lda	#3		KE	3770 error	dec	\$0620		;error counter
NE	3210	sta	\$31		NN	3780	beq	next		
					DO	3790	jmp	start		
					JD	3800 next	ldy	#\$ff		

JA	3810	sty	\$51	
MM	3820	iny		
KB	3830	sty	\$50	
GJ	3840	jmp	\$f969	;main error handler
CG	3850	step	ldx \$1c00	;move head half a track
AP	3860	inx		
CN	3870	txa		
FB	3880	and	#\$03	
FF	3890	sta	\$13ff	
DO	3900	lda	\$1c00	
JG	3910	and	#\$fc	
EI	3920	ora	\$13ff	
PD	3930	sta	\$1c00	
LP	3940	ldy	#\$05	
HI	3950	ldx	#\$ff	
NL	3960	loop1	dex	;allow head to settle
ME	3970	bne	loop1	
HE	3980	dey		
AG	3990	bne	loop1	
MI	4000	rts		

MJ	100	open1,8,1,"0:header.obj	
NB	105	rem save "@0:header.pal",8	
OO	110	sys700	;pal 64 assembler
OO	120	;	
EJ	130	this program lives inside the 1541	
IH	140	and creates an extra header block	
IK	150	it should be used with its basic	
AO	160	counterpart in the c64	
AC	170	;	
CM	180	.opt o1	
MK	190	*	= \$0800
LG	200	jmp	\$eaa0 ;reset disk drive
CF	210	lda	\$13fe ;track #;reentry from basic
AL	220	sta	\$18
LF	230	lda	\$13ff ;sector #
HM	240	sta	\$19
LA	250	lda	#\$00 ;checksum for header
LM	260	eor	\$16
HN	270	eor	\$17
DO	280	eor	\$18
PO	290	eor	\$19
LB	300	sta	\$1a
MI	310	jsr	\$f934 ;convert header to gcr
PL	320	ldx	#\$00
DH	330	lda	\$24,x ;store our header
KI	340	sta	\$1000,x ;image in a safe place
KD	350	inx	
ND	360	cpx	#\$0a
MD	370	bne	loop1
EP	380	lda	\$13fe ;set up for reading
GP	390	sta	\$06 ;last sector on this track
BN	400	ldx	\$13ff
BF	410	dex	
PM	420	stx	\$07
BN	430	lda	#\$00
EL	440	sta	\$3d
LI	450	sta	\$33
MI	460	sta	\$30

BB	470	lda	#\$06	
GK	480	sta	\$32	
JB	490	lda	#\$03	
HL	500	sta	\$31	
BK	510	jsr	\$f510	;read in header block
BG	520	jsr	\$f556	;wait for sync
KL	530	bvc	wait1	;read in data block
ON	540	clv		
OM	550	lda	\$1c01	
LC	560	sta	(\$30),y	
KB	570	iny		
KL	580	bne	wait1	
LE	590	ldy	#\$ba	
AO	600	bvc	wait2	
EC	610	clv		
EB	620	lda	\$1c01	
GI	630	sta	\$0100,y	
AG	640	iny		
CA	650	bne	wait2	
GF	660	clv		
EL	670	bvc	hold	;see if tail gap is
LK	680	clv		;over
KF	690	lda	\$1c01	
HE	700	cmp	#\$55	
HD	710	beq	hold	
KI	720	lda	\$1c0c	
OO	730	and	#\$1f	
NM	740	ora	#\$c0	
GO	750	sta	\$1c0c	
FL	760	lda	#\$ff	
KO	770	sta	\$1c03	
HB	780	lda	#\$ff	;write out 5 sync markers
MP	790	sta	\$1c01	
DL	800	ldx	#\$05	
EL	810	bvc	wait3	
GP	820	clv		
FP	830	dex		
CM	840	bne	wait3	
JM	850	ldx	#\$00	;write out extra header
JE	860	bvc	wait4	;block that was
KN	870	clv		;saved at \$1000
OD	880	lda	\$1000,x	
AG	890	sta	\$1c01	
AG	900	inx		
DG	910	cpx	#\$0a	
EB	920	bne	wait4	
PN	930	ldx	#\$09	;write out header gap
KD	940	bvc	wait5	
IH	950	clv		
GA	960	lda	#\$55	
AL	970	sta	\$1c01	
LI	980	dex		
MF	990	bne	wait5	
EI	1000	jsr	\$fe00	;terminate write mode
MN	1010	lda	#\$01	;jump to error handler
CC	1020	jmp	\$f969	;with a job all ok

Assembly Language Disk Access

Richard Evers, Editor

. . .it's no more difficult than Basic, once you try it. . .

For many programming applications, disk accessing through Basic is sufficient. At times, though, the speed of Basic disk access tends to induce a neanderthal reaction within us. This transformation is one in which we all take on from time to time, and as such, it is also one that is best left hidden. The solution: write the disk access routines in assembler.

There is a stigma attached to working with the drive at a machine code level. Programmers from all walks of life will often produce code destined for greatness, yet these same programmers will shy away from taking on the drive. The 'perhaps it will bite' syndrome prevails. This article has been written to dispel all such fears and show that the rumours have been more hysteria than fact. Enough babbling, time to produce.

If we were to write a simple file read routine in Basic, it would go something like this;

```
100 input "filename ";f$
110 open 7,8,9,(f$)
120 get#7,a$: print a$; if st=0 then 120
130 close 7
```

Not too difficult. Well, doing it in assembler is a bit more tedious, but easy to grasp after performing it a few times. Below is the flow of operations for the code written for this occasion.

1. Input filename from keyboard
2. Determine length of filename
3. Set-up parameters for the filename
4. Set-up logical address, secondary address, and device number
5. Open the file
6. Set input device as the file to be read
7. Read a character from disk and print it to the screen
8. Check the file status – if not the end of file, get another character (7)
9. Clear channel, close the file, and return to Basic

This is a very simple example of a sequential file read routine.

Notes On Direct Mode

If you were to assemble the source below and try to use it from direct mode, you will generate a syntax error on return to Basic. Why? Because direct mode statements are sent to the Basic input buffer for use. When a SYS statement is performed, the return address is placed on the stack, to be used when return to Basic is desired. Now the problems develop. In direct mode the SYS command will generate a return address that points right into the Basic input buffer. My examples use an Input routine to capture the desired filename. Input also uses the Basic input buffer. When the RTS is executed to return to Basic, it will land back into a pile of data it cannot understand, the filename. Instant syntax error. In program mode this will not happen since the return address will point into Basic text.

To get around this problem, a few methods exist. The first is to clear out the input buffer with zeros before returning to Basic. The next method is to retain the input buffer contents on entrance to the routine, and swap it back into the buffer before exit. A third method is to jump to a Basic warm start on exit instead of the RTS. A warm start will bring you cleanly back into direct mode Basic, irregardless of the mode in which you were working prior, direct or program mode. A thought to remember.

Procedure

In the first block of source, written for the Basic 4.0 Pet/CBM machines, you will notice that it looks a tad awkward in relation to the second block of source, written for the C64. The reason is that Commodore grew wiser, as far as programming goes, as time went by. They used their noggins and produced some really nice kernal routines to help make writing easier. In fact, my C64 source is virtually perfect for the VIC 20, with the exception of two ROM routines. The first routine, INPUT, is the Input routine, of which the address is located in the source. The second, PRTSTR, is a routine that will print a string of characters pointed to by the Y register (high byte), and the Accumulator (low byte). This routine will continue printing characters until a zero byte terminator is found. It's address can also be found listed in the source.

When you get a little better acquainted with the techniques involved, I am sure that you will be able to find quite a few shortcuts to take to produce the same effect. One shortcut is to use a Logical and Secondary address of 8. Instead of;

```
lda #7
sta logadd ; set logical address
lda #8
sta devnum ; set device number
lda #9
sta secadd ; set secondary address
```

for the Pet/CBM, you could simply:

```
lda #8 ; la, sa, devnum
sta logadd
sta devnum
sta secadd
```

It would save all of four bytes. For the C64 version, the code:

```
lda #7 ; logical address
ldx #8 ; device number
ldy #9 ; secondary address
```

could be optimized into:

```
lda #8 ; la, sa, devnum
tax
tay
```

Saving all of two bytes. If you are willing to stoop to a level of lack of clarity and further lack of adaptability, you can save a byte here and there. But wait until you understand the code properly.

Another note that might bother you. I used a very simple method to retrieve the filename, an Input statement. If you have been reading our magazine for any length of time, you might have noticed a few really sharp routines covered in prior issues. My favourite is the method of parsing the filename as a string following the SYS address when entering the routine. For purposes of simplicity, my example shows the Input statement. For your own creations, try to find one that suits the occasion best.

Very little more requires to be said about the routine that follows. It is a simple file read routine and performs no major miracles. Perhaps, though, you want to write to a file. This might be a tad more difficult, depending on the file type desired.

The easiest would be to create a Program file on disk. To do this, change the Secondary Address used in our example to a 1. With a Secondary Address of 1, all file work automatically defaults to a write, with the created file becoming a PRG type.

As you can probably guess, this feature was incorporated to make the Saving of Basic programs a little easier. Any Secondary Address greater than 1 and less than 15 will read from a file on default.

Once you have created a PRG write file, you have to be able to write to it. The code below shows the normal flow of operations. Below that is the changes required to make be able to write:

- Before -

```
ldx logadd ; get logical address
jsr chkin ; set input device
;
; ... get and print chars from file ...
more = *
jsr chrin ; get a char from disk
jsr chrout ; print it to the screen
```

- After -

```
ldx logadd ; get logical address
jsr chkout ; set output device
;
; ... get and print chars to file ...
more = *
jsr getchr ; ** your routine to supply the data
jsr chrout ; output to disk
```

Pretty easy. Instead of setting the input device (CHKIN), you now set the output device (CHKOUT). The Kernal address for CHKOUT is \$FFC9. Time for a quick interjection. If you want to simultaneously read from one file and write to another, there is one small problem. You would have to set the input device, read the character, set the output device, write the character, and continue this process until the input file has been read to your satisfaction. See below ;

```
ldx #readla ; read logical address
jsr chkin ; set input device
jsr chrin ; get a character
pha ; retain the byte
ldx #wrtla ; write logical address
jsr chkout ; set output device
pla ; get it back
jsr chrout ; output char
```

A little more tedious, but it works. Time to move on to file formats other than the PRG type.

Writing to a Sequential or User type file is a bit more difficult. A slight addition will have to be made to the filename, plus an alteration of the filename length will also have to be made to reflect this. The altered code could be something like this:


```

doit = *
ldy #0
;
; ... add write suffix to filename ...
alter = *
lda write,y ; get data to add
sta inbuf,x ; place it following the filename
inx
iny
cpy #length ; end of extra data
bne alter ; not yet
;
stx lennam ; set filename length

```

The ",s,w" at label WRITE should be placed near the data for the label PROMPT, just to keep it all in one place and out of the way. See below for label WRITE:

```

write = *
.asc ",s,w" ; change s to u for a user file
length = *-write

```

This covers writing to a Program, Sequential, and User file. As far as Relative files go, the modifications to my source would be a little more difficult. The Opening of a Relative file wouldn't be too horrific, but the positioning of records, writing for the first time, plus reading and writing thereafter could be code consuming. If you feel that life is not complete without this extra bit of knowledge, drop me a letter. With more than a couple requests, maybe.

As a final closing paragraph, I think a brief discussion regarding Opening files without a filename is in order. You may want to do this from time to time, as in the case of Opening the drives command channel for direct access work. The trick to this is simple. Don't bother with the filename pointers, and set the filename length to zero. If it was not a disk channel but a printer that required Opening, similar to the Basic incantation "OPEN 4,4", then assigning a value of 255 to the secondary address would perform the service. Two fast solutions for a few occasional problems. And so, with this article complete, I wish you luck in the land of disk access. With patience, and your thoughts collected, difficulties should not arise.

Basic 4.0 – Pet/CBM Version

MA	100	rem save "0:disk 1.pal",8	
HO	110	rem ** rte/85 – disk access in assembler – basic 4.0	
		pet/cbm **	
FI	120	open 4,8,1,"0:disk 1.obj"	
PB	130	sys(0)	
DK	140	.opt o4	
BN	150	* = \$027a	
GB	160	;	
MA	170	status = \$96 ;file status	
GE	180	lennam = \$d1 ;length of filename	
JK	190	logadd = \$d2 ;logical address	
CK	200	secadd = \$d3 ;secondary address	
DE	210	devnum= \$d4 ;device number	
GL	220	name = \$da ;filename (ptr)	
IF	230	inbuf = \$0200 ;input buffer	
NK	240	input = \$b4e2 ;input routine	
CG	250	prtstr = \$bb1d ;print string from y/a (hi/lo)	
CH	260	close = \$f2e2 ;close file	
FF	270	open = \$f563 ;open file	
BL	280	chkin = \$ffc6 ;open channel for input	(kernal)
KO	290	clrchn = \$ffcc ;restore default i/o	(kernal)
FI	300	chrin = \$ffcf ;input char from channel	(kernal)
BF	310	chrout = \$ffd2 ;output char to channel	(kernal)
GL	320	;	
DM	330	** disk read routine **	
KM	340	;	
LM	350	... ask for filename ...	
GB	360	ldy #>prompt	
EM	370	lda #<prompt	
LI	380	jsr prtstr ;print string	
CD	390	jsr input ;input filename into input buffer	
HN	400	ldx #0	
AB	410	;	
HL	420	... determine length of filename ...	
NK	430	getlen = *	
OA	440	lda inbuf,x ;get char from inbuf	
FD	450	beq doit ;if = 0 then end of filename	
IK	460	inx	
MD	470	bne getlen ;un-conditional branch	
GF	480	;	
LJ	490	... set-up filename ...	
JE	500	doit = *	
LJ	510	stx lennam ;set filename length	
NK	520	lda #<inbuf ;set ptrs to name	
DI	530	sta name	
HA	540	lda #>inbuf	
DK	550	sta name + 1	
GK	560	;	
DD	570	... set-up file for open ...	
ND	580	lda #7	
EP	590	sta logadd ;set logical address	
DF	600	lda #8	
OF	610	sta devnum ;set device number	


```

JG 620      lda  #9
LO 630      sta  secadd    ;set secondary address
HM 640      ldx  #0
AA 650 ;
BA 660 ; ... set length then open ...
NC 670      jsr  open      ;open file
HN 680      ldx  logadd    ;get logical address
DD 690      jsr  chkin     ;set input device
CD 700 ;
FB 710 ; ... get and print chars from file ...
IJ 720 more  =  *
LC 730      jsr  chrin     ;get a char from disk
HL 740      jsr  chrout    ;print it to the screen
DB 750      lda  status    ;check if end-of-file
GK 760      beq  more      ;not yet - go for some more
IH 770 ;
BM 780      jsr  clrchn    ;clear channel
ED 790      lda  logadd    ;logical address
DL 800      jsr  close     ;close file
PC 810      rts           ;return to basic
KK 820 ;
NG 830 prompt =  *
MJ 840      .asc  "filename ? "
MI 850      .byt 0
CN 860 ;
CE 870 .end

```

Commodore 64 Version

```

OA 100 rem save "0:disk 2.pal",8
CN 110 rem ** rte/85 - disk access in assembler
    - c64 version **
JI 120 open 4,8,1, "0:disk 2.obj"
NB 130 sys(700)
DK 140 .opt o4
PM 150 *      = $033c
GB 160 ;
MB 170 inbuf = $0200 ;input buffer
PI 180 input  = $a560 ;input routine      ($c560 for vic)
KJ 190 prtstr = $ab1e ;print string from y/a ($cb1e for vic)
CH 200 readst = $ffb7 ;read file status      (kernal)
FP 210 setlsf = $ffb8 ;set la, sa, and devnum (kernal)
PG 220 setnam = $ffbd ;set-up filename      (kernal)
DI 230 open   = $ffc0 ;open file            (kernal)
NJ 240 close  = $ffc3 ;close file           (kernal)
DJ 250 chkin  = $ffc6 ;open channel for input (kernal)
MM 260 clrchn = $ffcc ;restore default i/o   (kernal)
HG 270 chrin  = $ffcf ;input char from channel (kernal)
DD 280 chrout = $ffd2 ;output char to channel (kernal)

```

```

IJ 290 ;
FK 300 ; ** disk read routine **
MK 310 ;
NK 320 ; ... ask for filename ...
IP 330      ldy  #>prompt
GK 340      lda  #<prompt
NG 350      jsr  prtstr    ;print string
EB 360      jsr  input     ;input filename into input
                                buffer
JL 370      ldx  #0
CP 380 ;
JJ 390 ; ... determine length of filename ...
PI 400 getlen =  *
AP 410      lda  inbuf,x   ;get char from inbuf
HB 420      beq  doit      ;if = 0 then end of filename
KI 430      inx
OB 440      bne  getlen    ;un-conditional branch
ID 450 ;
NH 460 ; ... set-up filename ...
LC 470 doit  =  *
DC 480      txa           ;a = length
EN 490      ldx  #<inbuf  ;x = name ptr lo
LM 500      ldy  #>inbuf  ;y = name ptr hi
LG 510      jsr  setnam    ;set-up filename
OH 520 ;
DK 530 ; ... set-up file then open ...
JD 540      lda  #7       ;logical address
MO 550      ldx  #8       ;device number
LN 560      ldy  #9       ;secondary address
LE 570      jsr  setlsf    ;set la, sa, and devnum
DN 580      jsr  open      ;open file
HM 590      ldx  #7       ;logical address
JN 600      jsr  chkin     ;set input device
IN 610 ;
LL 620 ; ... get and print chars from file ...
OD 630 more  =  *
BN 640      jsr  chrin     ;get a char from disk
NF 650      jsr  chrout    ;print it to the screen
NF 660      jsr  readst    ;check if end-of-file
ME 670      beq  more      ;not yet - go for some more
OB 680 ;
HG 690      jsr  clrchn    ;clear channel
JN 700      lda  #7       ;logical address
PH 710      jsr  close     ;close the file
ML 720      rts
AF 730 ;
DB 740 prompt =  *
CE 750      .asc  "filename ? "
CD 760      .byt 0
IH 770 ;
IO 780 .end

```


Directory Match

Pierre Corriveau
Montreal, Quebec

As time passes by, your library of programs grows bigger and bigger. Utility programs, games, educational programs, and a lot of them have similar names.

Then one evening, you need to find that one utility program, you don't remember the whole name but you know there was the word "test" in it. Was it "Prt test pgm", "Test printer" or "Print. test"? Which disk was it on?

There you go, looking through all your disks, typing LOAD "\$", 8 and LIST dozens of times. Pattern matching can't help you since you don't know where the word "test" is located in the filename. You have to examine each filename; a long and boring job.

That's why DIR MATCH was created. Now all you have to do is load it in, insert the disk you want to search in the drive and type SYS 49152, "test". DIR MATCH will list any filename that contains "test" in it. You can also type SYS 49152 alone and it will list the whole directory on screen. To stop the listing, hold the SHIFT key. To abort it, press the BACK ARROW key.

Also, the program is made so that it can be relocated. The basic loader will take care of that. The PAL source code is included for those who are tuned to the fast paced world of machine language.

But wait a minute, what if you also forgot which disk DIR MATCH was on ?!?

DIR MATCH Basic Loader

```
CP 100 rem the routine uses channels #13 and #15
CN 110 ad = 49152:rem change if you wish
JH 120 print "r directory search routine"
NC 130 print "q sys" ad = directory listing"
MB 140 print "q sys" ad ", " chr$(34) "mon" chr$(34)
OJ 150 print "will list filenames containing"
PN 160 print "'mon' like 'micromon' and '64mon'"
CA 170 print "q sys" ad ", a$ also works"
AB 180 print "q pause listing with shift"
MO 190 print "halt with run/stop"
NI 200 print "q relocatable: look at line 110"
AE 210 print "...please wait. . ."
AJ 220 gosub 250
GO 230 end
EG 240 :
JO 250 a = ad:rem change if you wish
AM 260 read d:if d = -1 then 280
PL 270 poke a,d:ck = ck + d:a = a + 1:goto 260
HO 280 if ck <> 48226 then print "error in data lines 390-840":end
CA 290 a = ad
BK 300 for i = 1 to 11
HF 310 read rt:ch = ch + rt
OA 320 read of:if of = -1 then 360
DM 330 ch = ch + of
CE 340 poke a + of + 1,int((a + rt)/256):poke a + of,(a + rt)-256*int((a + rt)/256)
BA 350 goto 320
MG 360 next
FL 370 if ch <> 8490 then print "error in data lines 870-920":end
IJ 380 return
KH 390 data 169, 0, 141, 75, 193, 32, 121, 0
NA 400 data 240, 47, 32, 253, 174, 32, 158, 173
AE 410 data 32, 143, 173, 169, 13, 32, 210, 255
CF 420 data 160, 0, 177, 100, 201, 17, 144, 5
HP 430 data 162, 23, 76, 58, 164, 141, 75, 193
CI 440 data 200, 177, 100, 133, 20, 200, 177, 100
HO 450 data 133, 21, 165, 100, 164, 101, 32, 219
MO 460 data 182, 169, 0, 32, 189, 255, 169, 15
EN 470 data 162, 8, 160, 15, 32, 186, 255, 32
PC 480 data 192, 255, 144, 8, 72, 32, 232, 192
HC 490 data 104, 76, 59, 164, 169, 1, 162, 205
IH 500 data 160, 235, 32, 189, 255, 169, 13, 162
EE 510 data 8, 160, 0, 32, 186, 255, 32, 192
HF 520 data 255, 176, 225, 162, 15, 32, 198, 255
KC 530 data 32, 207, 255, 201, 48, 240, 3, 76
DM 540 data 220, 192, 32, 204, 255, 162, 13, 32
EI 550 data 198, 255, 32, 207, 255, 32, 207, 255
JF 560 data 32, 204, 255, 32, 246, 192, 32, 47
OJ 570 data 193, 32, 225, 255, 240, 8, 32, 246
PL 580 data 192, 16, 6, 32, 47, 193, 76, 232
KC 590 data 192, 173, 75, 193, 208, 7, 173, 141
DE 600 data 2, 208, 251, 240, 225, 162, 0, 189
DJ 610 data 80, 193, 232, 201, 34, 208, 248, 142
LN 620 data 77, 193, 160, 0, 189, 80, 193, 201
BK 630 data 34, 240, 206, 209, 20, 240, 8, 238
```



```

NN 640 data 77, 193, 174, 77, 193, 208, 235, 200
PD 650 data 232, 204, 75, 193, 208, 230, 32, 47
LO 660 data 193, 76, 145, 192, 32, 210, 255, 165
IA 670 data 144, 208, 5, 32, 207, 255, 208, 244
KF 680 data 32, 204, 255, 169, 13, 32, 195, 255
AF 690 data 169, 15, 32, 195, 255, 96, 169, 255
CH 700 data 141, 76, 193, 162, 13, 32, 198, 255
HE 710 data 32, 207, 255, 32, 207, 255, 240, 32
DI 720 data 32, 207, 255, 141, 78, 193, 32, 207
ME 730 data 255, 141, 79, 193, 160, 255, 200, 32
NI 740 data 207, 255, 153, 80, 193, 240, 9, 201
FD 750 data 34, 208, 243, 141, 76, 193, 240, 238
FL 760 data 32, 204, 255, 173, 76, 193, 96, 174
BO 770 data 78, 193, 173, 79, 193, 32, 205, 189
CB 780 data 160, 255, 169, 32, 32, 210, 255, 200
DB 790 data 185, 80, 193, 208, 247, 169, 13, 32
MK 800 data 210, 255, 96, 0, 255, 0, 32, 32
MC 810 data 32, 32, 32, 32, 32, 32, 32, 32
GD 820 data 32, 32, 32, 32, 32, 32, 32, 32
AE 830 data 32, 32, 32, 32, 32, 32, 32, 32
DE 840 data 32, 32, 32, 32, -1
GM 850 :
HA 860 rem relocation code
HD 870 data 331, 3, 38, 162, 210, -1, 220, 120
EG 880 data -1, 246, 140, 151, -1, 303, 143, 156
PA 890 data 215, -1, 232, 159, 78, -1, 336, 176
NI 900 data 189, 283, 321, -1, 333, 184, 200, 203
KI 910 data -1, 145, 218, -1, 332, 249, 292, 300
MP 920 data -1, 334, 268, 304, -1, 335, 274, 307, -1

```

DIR MATCH PAL Source Code

```

FD 100 sys700
LN 110 ;print the disk directory
OO 120 ;
KO 130 .opt oo
NH 140 * = $c000
FM 150 setfls = $ffbba
CL 160 setnam = $ffbd
JO 170 open = $ffc0
MD 180 chkin = $ffc6
CK 190 chrin = $ffcf
EL 200 clrchn = $ffcc
PB 210 close = $ffc3
DA 220 chrout = $ffd2
BB 230 stop = $ffe1
AI 240 st = $90
CG 250 chrgot = $79
CJ 260 string = $14
EI 270 ;
CB 280 ;check if param is present
NA 290 lda #0
PM 300 sta strlen
EF 310 jsr chrgot
AL 320 beq opencmd
AM 330 ;
CA 340 ;get the string
PO 350 jsr $aefd ;check comma
BF 360 jsr $ad9e ;get parameter
HA 370 jsr $ad8f ;check if string
CA 380 lda #13
NF 390 jsr chrout
LN 400 ldy #0
EH 410 lda ($64),y ;get string length

```

```

OF 420 cmp #17 ;if > 16 error
NA 430 bcc okstrlen
OP 440 ldx #23 ;string too long error
MM 450 jmp $a43a
ME 460 okstrlen = *
FC 470 sta strlen ;save length
AM 480 iny
EB 490 lda ($64),y ;save pointer to string
LI 500 sta string
ON 510 iny
ON 520 lda ($64),y
LG 530 sta string+1
AL 540 lda $64
NB 550 ldy $65
PP 560 jsr $b6db ;clean descriptor stack
AL 570 ;
OC 580 ;set command channel for open
HN 590 opencmd = *
DE 600 lda #0
DC 610 jsr setnam
IP 620 lda #15
NM 630 ldx #8
MG 640 ldy #15
CE 650 jsr setfls
IA 660 jsr open
MK 670 bcc noerror
OB 680 ;
AM 690 knerror = *
KI 700 pha ;save error number
JL 710 jsr fini ;close channel #15
IF 720 pla
MJ 730 jmp $a43b ;print kernal error msg
KF 740 ;
HE 750 ;set channel #13 for open
PA 760 noerror = *
PO 770 lda #1
DA 780 ldx #$cd ;pt to '$' of keybrd mat.
AC 790 ldy #$eb
BO 800 jsr setnam
AK 810 ;
EE 820 ;set file name to "$"
EM 830 lda #13
PJ 840 ldx #8
NJ 850 ldy #0
EB 860 jsr setfls
KN 870 jsr open
JL 880 bcs knerror
AP 890 ;
KP 900 ;check if everything ok
GH 910 ldx #15
DJ 920 jsr chkin
CL 930 jsr chrin
BD 940 cmp #"0" ;'0' means ok
GM 950 beq br1
KH 960 jmp perror ;print error and exit
AE 970 ;
EB 980 ;looking good. disconnect ch #15
ME 990 ;and start printing
OF 1000 ;
KD 1010 br1 = *
DA 1020 jsr clrchn
IO 1030 ldx #13
LA 1040 jsr chkin
ME 1050 jsr chrin ;skip load adrs
LK 1060 jsr chrin ;with 2 chrins
FD 1070 jsr clrchn

```


OK	1080 ;			CF	1740	lda	#13	
FE	1090 ;print disk name			LE	1750	jsr	close	
OJ	1100	jsr	getfile	MG	1760	lda	#15	
LA	1110 dir	=	*	PF	1770	jsr	close	
NO	1120	jsr	prtfile	AO	1780	rts		
AA	1130 nextfile	=	*	EH	1790 ;			
HJ	1140	jsr	stop	NM	1800 ;input filename in buffer			
OE	1150	beq	done	ON	1810 ;on return flag = 255 means end			
KN	1160	jsr	getfile	DP	1820 ;	flag =	34 means ok	
NC	1170	bpl	notdone	MJ	1830 ;			
CN	1180	jsr	prtfile	EA	1840 getfile	=	*	
PP	1190 done	=	*	MJ	1850	lda	#255	
EJ	1200	jmp	fini	HK	1860	sta	flag	
HM	1210 notdone	=	*	AD	1870	ldx	#13	
GO	1220	lda	strlen	DF	1880	jsr	chkin	
PG	1230	bne	match	PG	1890	jsr	chrin	;skip forward chain
JM	1240 waitshft	=	*	ME	1900	jsr	chrin	;except zero chain
DF	1250	lda	\$028d	EH	1910	beq	get2	;means end
KO	1260	bne	waitshft	GP	1920 ;			
JF	1270	beq	dir	EI	1930 cont	=	*	
GH	1280 ;			BC	1940	jsr	chrin	;get nb of blocks
KN	1290 ;check for a match			HO	1950	sta	fname	
KF	1300 match	=	*	IL	1960	jsr	chrin	
GK	1310	ldx	#0	NL	1970	sta	fname + 1	
CL	1320 match2	=	*	CD	1980 ;			
PN	1330	lda	fname + 2,x	FB	1990 ;let's get text			
IB	1340	inx		CJ	2000	ldy	#255	
IB	1350	cmp	#34	NG	2010 text	=	*	
FF	1360	bne	match2	EM	2020	iny		
CI	1370	stx	flindex	OP	2030	jsr	chrin	
LO	1380 match1	=	*	FO	2040	sta	fname + 2,y	
LD	1390	ldy	#0	CK	2050	beq	get2	;0 = end of line
OO	1400 ;			IE	2060	cmp	#34	;quote " ?
KB	1410 nextchar	=	*	NK	2070	bne	text	;no, get some more
JD	1420	lda	fname + 2,x	KO	2080	sta	flag	;yes, set the flag
EF	1430	cmp	#34	JE	2090	beq	text	;jump
FJ	1440	beq	nextfile	AI	2100 get2	=	*	
KK	1450	cmp	(string),y	FE	2110	jsr	clrchn	
EP	1460	beq	charmat	NG	2120	lda	flag	
CH	1470	inc	flindex	OD	2130	rts		
DF	1480	ldx	flindex	CN	2140 ;			
NI	1490	bne	match1	JJ	2150 ;routine to print filename			
HE	1500 charmat	=	*	IC	2160 ;filename ends with a zero byte			
MO	1510	iny		KD	2170 prtfile	=	*	
MM	1520	inx		LJ	2180	ldx	fname	;print nb of blocks
FE	1530	cpy	strlen	LF	2190	lda	fname + 1	
FE	1540	bne	nextchar	BD	2200	jsr	\$bdcd	
EI	1550 ;			EG	2210	ldy	#255	
DG	1560 ;got a match !			FG	2220	lda	# " "	;1 space
PK	1570	jsr	prtfile	FD	2230 prt1	=	*	
FE	1580	jmp	nextfile	HJ	2240	jsr	chrout	
MK	1590 ;			KK	2250	iny		
PP	1600 ;there was an error; print it			DI	2260	lda	fname + 2,y	
AI	1610 ;the first char of the error msg			EB	2270	bne	prt1	
NP	1620 ;is already in the accumulator			OG	2280	lda	#13	
EN	1630 ;			JM	2290	jsr	chrout	
HJ	1640 perror	=	*	IO	2300	rts		
JE	1650	jsr	chrout	MH	2310 ;			
PN	1660	lda	st	BL	2320 strlen	.byte	0	
EA	1670	bne	fini	IN	2330 flag	.byte	255	
AK	1680	jsr	chrin	PG	2340 flindex	.byte	0	
JH	1690	bne	pperror	CE	2350 fname	.asc	"	;30 spaces
KB	1700 ;							
LN	1710 ;all done. go back to basic							
IA	1720 fini	=	*					
JM	1730	jsr	clrchn					

Commodore 64 Memory Configurations

William Levak
Ann Arbor, MI

Effects Of The PLA Chip In All Situations

The major obstacle to understanding the Commodore 64's memory is the 82S100 programmed logic array (PLA) that controls it. What we need is the truth table for the chip. A truth table is a table that shows the output on every output pin for every possible combination of inputs on the input pins. We would normally find the truth table for a chip on the manufacturer's data sheet. However, a programmed logic array does not have a truth table until it is programmed (in this case by Commodore), and Commodore has not published the truth table. I have therefore read out the chip and obtained the truth table which is shown in the first table.

How To Read The Truth Table

At the left side of the table are the names of the inputs (pins I0-I15) and outputs (pins F0-F7) of the PLA as given in the circuit diagram in the Commodore 64 Programmer's Reference Guide. The 26 columns in the table each represent a combination of inputs recognized by the PLA and the outputs that combination causes. The first column with all the inputs marked with a '-' represents the default output state, that is, if none of the other input conditions are met, then the output will be as indicated in this column. For all the other columns, a '1' indicates a high input state (+5 V), a '0' indicates a low input state (0 V), and an 'x' represents an input that is ignored for this column. If the input conditions in a particular column are met then the outputs will be changed from the default condition.

A "1" indicates that the output will change from low to high,
a "0" indicates that the output will change from high to low,
and
an "x" indicates that the output will remain in the default state.

It is possible that more than one input condition will be met at the same time, and their outputs will then be combined. This occurs only with certain timing signals necessary to access RAM.

Although the truth table is the most complete description of the PLA possible, it is not very useful to the average programmer. I have therefore produced memory maps of the fourteen possible memory configurations of the Commodore 64. The memory configuration is selected by five lines on the PLA:

LORAM, HIRAM, and CHAREN are controlled by bits 0, 1, and 2 at memory location 1.

GAME and EXROM are two lines on the cartridge expansion port and are controlled by the circuitry on the cartridge.

All five lines are normally high.

The addresses at the left of the table are the beginning to ending addresses of each 4K block. KERNL, BASIC, and CHARA refer to the kernal, BASIC and character ROMs respectively. I/O refers to the 4K I/O block which contains all the I/O chips and the color memory. ROM H and ROM L refer to two cartridge ROMs which can be up to 8K in length. The 4K ROM H blocks in the VIC column in the last configuration (the game configuration) are the top 4K of the 8K ROM H block seen by the CPU. Ram, of course, refers to random access memory.

Each memory map contains four columns. This is because the memory configuration differs according to the device accessing it and whether it is reading or writing. The column on the right is what the VIC chip sees when reading. The column on the left is what the CPU sees when writing. Both columns in the center are what the CPU sees when reading, but under different conditions. The VIC chip and the CPU share the same address and data busses, but on opposite phases of the clock cycle. Occasionally the VIC chip needs the busses longer than half a clock cycle. When this occurs, the VIC chip pulls line BA low. When the CPU sees that BA is low it runs for three clock cycles and then halts. During these three cycles the memory configuration is as described in the column marked R(-BA). At all other times it is as described in the column headed R(BA). Notice that this affects only the CPU reading the I/O block. During these three cycles the I/O block is unavailable and RAM is seen instead. Writing to the I/O block is still possible however.

Some final notes: In compiling this information I have read out three PLAs and each had different numbers on them. In order to eliminate transcription errors, all the tables were generated directly by computer programs from a disk file containing the information from the PLA. The careful reader will spot differences between the information presented here and that contained in the Commodore 64 Programmer's Reference Guide. All such differences were verified manually against the original data file created by reading out the PLA, which itself was verified against two other PLAs as noted above.

Inputs		Commodore 64 PLA																												
I0	CAS	x	x	x	x	x	x	x	x	x	x	0	x	x	x	x	x	x	x	-	1	0	0	0	0					
I1	LORAM	x	x	x	x	1	x	x	1	x	x	1	x	x	x	x	1	x	x	x	1	-	x	x	x	x	x			
I2	HIRAM	x	x	1	x	1	1	x	0	1	x	0	x	x	x	1	0	1	1	1	1	-	x	x	x	x	x			
I3	CHAREN	x	x	x	x	x	1	x	1	1	x	1	x	x	x	0	0	0	x	x	x	-	x	x	x	x	x			
I4	VA14	x	x	x	x	x	x	x	x	x	x	x	x	1	1	x	x	x	x	x	x	-	x	x	x	x	x			
I5	A15	x	1	1	1	1	1	1	1	1	1	1	1	x	x	1	1	1	1	1	1	-	x	x	x	x	x			
I6	A14	x	1	0	0	0	1	1	1	1	1	1	1	x	x	1	1	1	1	1	0	-	x	1	0	0	1			
I7	A13	x	1	1	0	0	0	0	0	0	0	0	0	x	x	0	0	0	1	1	1	-	x	x	1	0	0			
I8	A12	x	x	x	x	x	1	1	1	1	1	1	1	x	x	1	1	1	x	x	x	-	x	x	x	1	0			
I9	BA	x	x	x	x	x	x	x	x	1	1	1	x	x	x	x	x	x	x	x	x	-	x	x	x	x	x			
I10	AEC	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	-	x	x	x	x	x			
I11	R/W	x	x	1	x	1	0	0	0	1	1	1	x	x	x	x	x	x	x	x	x	-	x	x	x	x	x			
I12	EXROM	1	1	0	1	0	x	1	x	x	1	x	x	0	1	x	x	0	x	0	x	-	x	1	1	1	1			
I13	GAME	0	0	0	0	x	x	0	x	x	0	x	x	x	1	1	1	0	1	0	1	-	x	0	0	0	0			
I14	VA13	1	x	x	x	x	x	x	x	x	x	x	x	0	0	x	x	x	x	x	x	-	x	x	x	x	x			
I15	VA12	1	x	x	x	x	x	x	x	x	x	x	x	1	1	x	x	x	x	x	x	-	x	x	x	x	x			
Outputs																														
F0	CASRAM	1	1	1	1	1	1	1	1	1	1	x	1	1	1	1	1	1	1	1	0	1	1	1	1	1				
F1	BASIC	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	1	x	x	x	x	x				
F2	KERNAL	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	x	1	x	x	x	x				
F3	CHAROM	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	x	x	x	1	x	x	x	x				
F4	GR/W	x	x	x	x	x	x	x	x	x	x	0	x	x	x	x	x	x	x	x	1	x	x	x	x	x				
F5	I/O	x	x	x	x	x	0	0	0	0	0	0	x	x	x	x	x	x	x	x	1	x	x	x	x	x				
F6	ROM L	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	x	x	x	x	x				
F7	ROM H	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	x	x	x	x	x				

GAME 1
 EXROM 1
 CHAREN 1
 HIRAM 1
 LORAM 1

	CPU			VIC
	W	R(BA)	R(BA)	R
\$F000 - FFFF	ram	KERNL	KERNL	ram
\$E000 - EFFF	ram	KERNL	KERNL	ram
\$D000 - DFFF	I/O	ram	I/O	ram
\$C000 - CFFF	ram	ram	ram	ram
\$B000 - BFFF	ram	BASIC	BASIC	ram
\$A000 - AFFF	ram	BASIC	BASIC	ram
\$9000 - 9FFF	ram	ram	ram	CHARA
\$8000 - 8FFF	ram	ram	ram	ram
\$7000 - 7FFF	ram	ram	ram	ram
\$6000 - 6FFF	ram	ram	ram	ram
\$5000 - 5FFF	ram	ram	ram	ram
\$4000 - 4FFF	ram	ram	ram	ram
\$3000 - 3FFF	ram	ram	ram	ram
\$2000 - 2FFF	ram	ram	ram	ram
\$1000 - 1FFF	ram	ram	ram	CHARA
\$0000 - 0FFF	ram	ram	ram	ram

GAME 1
 EXROM 1
 CHAREN 0
 HIRAM 1
 LORAM 1

	CPU			VIC
	W	R(BA)	R(BA)	R
\$F000 - FFFF	ram	KERNL	KERNL	ram
\$E000 - EFFF	ram	KERNL	KERNL	ram
\$D000 - DFFF	ram	CHARA	CHARA	ram
\$C000 - CFFF	ram	ram	ram	ram
\$B000 - BFFF	ram	BASIC	BASIC	ram
\$A000 - AFFF	ram	BASIC	BASIC	ram
\$9000 - 9FFF	ram	ram	ram	CHARA
\$8000 - 8FFF	ram	ram	ram	ram
\$7000 - 7FFF	ram	ram	ram	ram
\$6000 - 6FFF	ram	ram	ram	ram
\$5000 - 5FFF	ram	ram	ram	ram
\$4000 - 4FFF	ram	ram	ram	ram
\$3000 - 3FFF	ram	ram	ram	ram
\$2000 - 2FFF	ram	ram	ram	ram
\$1000 - 1FFF	ram	ram	ram	CHARA
\$0000 - 0FFF	ram	ram	ram	ram

GAME 1
 EXROM 0
 CHAREN 1
 HIRAM 1
 LORAM 1

GAME 1
 EXROM 0
 CHAREN 0
 HIRAM 1
 LORAM 1

	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 – FFFF	ram	KERNL	KERNL	ram
\$E000 – EFFF	ram	KERNL	KERNL	ram
\$D000 – DFFF	I/O	ram	I/O	ram
\$C000 – CFFF	ram	ram	ram	ram
\$B000 – BFFF	ram	BASIC	BASIC	ram
\$A000 – AFFF	ram	BASIC	BASIC	ram
\$9000 – 9FFF	ram	ROM L	ROM L	CHARA
\$8000 – 8FFF	ram	ROM L	ROM L	ram
\$7000 – 7FFF	ram	ram	ram	ram
\$6000 – 6FFF	ram	ram	ram	ram
\$5000 – 5FFF	ram	ram	ram	ram
\$4000 – 4FFF	ram	ram	ram	ram
\$3000 – 3FFF	ram	ram	ram	ram
\$2000 – 2FFF	ram	ram	ram	ram
\$1000 – 1FFF	ram	ram	ram	CHARA
\$0000 – 0FFF	ram	ram	ram	ram

	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 – FFFF	ram	KERNL	KERNL	ram
\$E000 – EFFF	ram	KERNL	KERNL	ram
\$D000 – DFFF	ram	CHARA	CHARA	ram
\$C000 – CFFF	ram	ram	ram	ram
\$B000 – BFFF	ram	BASIC	BASIC	ram
\$A000 – AFFF	ram	BASIC	BASIC	ram
\$9000 – 9FFF	ram	ROM L	ROM L	CHARA
\$8000 – 8FFF	ram	ROM L	ROM L	ram
\$7000 – 7FFF	ram	ram	ram	ram
\$6000 – 6FFF	ram	ram	ram	ram
\$5000 – 5FFF	ram	ram	ram	ram
\$4000 – 4FFF	ram	ram	ram	ram
\$3000 – 3FFF	ram	ram	ram	ram
\$2000 – 2FFF	ram	ram	ram	ram
\$1000 – 1FFF	ram	ram	ram	CHARA
\$0000 – 0FFF	ram	ram	ram	ram

GAME 0
 EXROM 0
 CHAREN 1
 HIRAM 1
 LORAM 1

GAME 0
 EXROM 0
 CHAREN 0
 HIRAM 1
 LORAM 1

	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 – FFFF	ram	KERNL	KERNL	ram
\$E000 – EFFF	ram	KERNL	KERNL	ram
\$D000 – DFFF	I/O	ram	I/O	ram
\$C000 – CFFF	ram	ram	ram	ram
\$B000 – BFFF	ram	ROM H	ROM H	ram
\$A000 – AFFF	ram	ROM H	ROM H	ram
\$9000 – 9FFF	ram	ROM L	ROM L	CHARA
\$8000 – 8FFF	ram	ROM L	ROM L	ram
\$7000 – 7FFF	ram	ram	ram	ram
\$6000 – 6FFF	ram	ram	ram	ram
\$5000 – 5FFF	ram	ram	ram	ram
\$4000 – 4FFF	ram	ram	ram	ram
\$3000 – 3FFF	ram	ram	ram	ram
\$2000 – 2FFF	ram	ram	ram	ram
\$1000 – 1FFF	ram	ram	ram	CHARA
\$0000 – 0FFF	ram	ram	ram	ram

	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 – FFFF	ram	KERNL	KERNL	ram
\$E000 – EFFF	ram	KERNL	KERNL	ram
\$D000 – DFFF	ram	CHARA	CHARA	ram
\$C000 – CFFF	ram	ram	ram	ram
\$B000 – BFFF	ram	ROM H	ROM H	ram
\$A000 – AFFF	ram	ROM H	ROM H	ram
\$9000 – 9FFF	ram	ROM L	ROM L	CHARA
\$8000 – 8FFF	ram	ROM L	ROM L	ram
\$7000 – 7FFF	ram	ram	ram	ram
\$6000 – 6FFF	ram	ram	ram	ram
\$5000 – 5FFF	ram	ram	ram	ram
\$4000 – 4FFF	ram	ram	ram	ram
\$3000 – 3FFF	ram	ram	ram	ram
\$2000 – 2FFF	ram	ram	ram	ram
\$1000 – 1FFF	ram	ram	ram	CHARA
\$0000 – 0FFF	ram	ram	ram	ram

GAME 0 EXROM 0 CHAREN 1 HIRAM 1 LORAM 0				
	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 – FFFF	ram	KERNL	KERNL	ram
\$E000 – EFFF	ram	KERNL	KERNL	ram
\$D000 – DFFF	I/O	ram	I/O	ram
\$C000 – CFFF	ram	ram	ram	ram
\$B000 – BFFF	ram	ROM H	ROM H	ram
\$A000 – AFFF	ram	ROM H	ROM H	ram
\$9000 – 9FFF	ram	ram	ram	CHARA
\$8000 – 8FFF	ram	ram	ram	ram
\$7000 – 7FFF	ram	ram	ram	ram
\$6000 – 6FFF	ram	ram	ram	ram
\$5000 – 5FFF	ram	ram	ram	ram
\$4000 – 4FFF	ram	ram	ram	ram
\$3000 – 3FFF	ram	ram	ram	ram
\$2000 – 2FFF	ram	ram	ram	ram
\$1000 – 1FFF	ram	ram	ram	CHARA
\$0000 – 0FFF	ram	ram	ram	ram

GAME 0 EXROM 0 CHAREN 0 HIRAM 1 LORAM 0				
	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 – FFFF	ram	KERNL	KERNL	ram
\$E000 – EFFF	ram	KERNL	KERNL	ram
\$D000 – DFFF	ram	CHARA	CHARA	ram
\$C000 – CFFF	ram	ram	ram	ram
\$B000 – BFFF	ram	ROM H	ROM H	ram
\$A000 – AFFF	ram	ROM H	ROM H	ram
\$9000 – 9FFF	ram	ram	ram	CHARA
\$8000 – 8FFF	ram	ram	ram	ram
\$7000 – 7FFF	ram	ram	ram	ram
\$6000 – 6FFF	ram	ram	ram	ram
\$5000 – 5FFF	ram	ram	ram	ram
\$4000 – 4FFF	ram	ram	ram	ram
\$3000 – 3FFF	ram	ram	ram	ram
\$2000 – 2FFF	ram	ram	ram	ram
\$1000 – 1FFF	ram	ram	ram	CHARA
\$0000 – 0FFF	ram	ram	ram	ram

GAME 1 EXROM x CHAREN 1 HIRAM 1 LORAM 0				
	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 – FFFF	ram	KERNL	KERNL	ram
\$E000 – EFFF	ram	KERNL	KERNL	ram
\$D000 – DFFF	I/O	ram	I/O	ram
\$C000 – CFFF	ram	ram	ram	ram
\$B000 – BFFF	ram	ram	ram	ram
\$A000 – AFFF	ram	ram	ram	ram
\$9000 – 9FFF	ram	ram	ram	CHARA
\$8000 – 8FFF	ram	ram	ram	ram
\$7000 – 7FFF	ram	ram	ram	ram
\$6000 – 6FFF	ram	ram	ram	ram
\$5000 – 5FFF	ram	ram	ram	ram
\$4000 – 4FFF	ram	ram	ram	ram
\$3000 – 3FFF	ram	ram	ram	ram
\$2000 – 2FFF	ram	ram	ram	ram
\$1000 – 1FFF	ram	ram	ram	CHARA
\$0000 – 0FFF	ram	ram	ram	ram

GAME 1 EXROM x CHAREN 0 HIRAM 1 LORAM 0				
	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 – FFFF	ram	KERNL	KERNL	ram
\$E000 – EFFF	ram	KERNL	KERNL	ram
\$D000 – DFFF	ram	CHARA	CHARA	ram
\$C000 – CFFF	ram	ram	ram	ram
\$B000 – BFFF	ram	ram	ram	ram
\$A000 – AFFF	ram	ram	ram	ram
\$9000 – 9FFF	ram	ram	ram	CHARA
\$8000 – 8FFF	ram	ram	ram	ram
\$7000 – 7FFF	ram	ram	ram	ram
\$6000 – 6FFF	ram	ram	ram	ram
\$5000 – 5FFF	ram	ram	ram	ram
\$4000 – 4FFF	ram	ram	ram	ram
\$3000 – 3FFF	ram	ram	ram	ram
\$2000 – 2FFF	ram	ram	ram	ram
\$1000 – 1FFF	ram	ram	ram	CHARA
\$0000 – 0FFF	ram	ram	ram	ram

GAME 1 0
EXROM x 0
CHAREN 1 1
HIRAM 0 0
LORAM 1 1

	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 - FFFF	ram	ram	ram	ram
\$E000 - EFFF	ram	ram	ram	ram
\$D000 - DFFF	I/O	ram	I/O	ram
\$C000 - CFFF	ram	ram	ram	ram
\$B000 - BFFF	ram	ram	ram	ram
\$A000 - AFFF	ram	ram	ram	ram
\$9000 - 9FFF	ram	ram	ram	CHARA
\$8000 - 8FFF	ram	ram	ram	ram
\$7000 - 7FFF	ram	ram	ram	ram
\$6000 - 6FFF	ram	ram	ram	ram
\$5000 - 5FFF	ram	ram	ram	ram
\$4000 - 4FFF	ram	ram	ram	ram
\$3000 - 3FFF	ram	ram	ram	ram
\$2000 - 2FFF	ram	ram	ram	ram
\$1000 - 1FFF	ram	ram	ram	CHARA
\$0000 - 0FFF	ram	ram	ram	ram

GAME 1
EXROM x
CHAREN 0
HIRAM 0
LORAM 1

	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 - FFFF	ram	ram	ram	ram
\$E000 - EFFF	ram	ram	ram	ram
\$D000 - DFFF	ram	CHARA	CHARA	ram
\$C000 - CFFF	ram	ram	ram	ram
\$B000 - BFFF	ram	ram	ram	ram
\$A000 - AFFF	ram	ram	ram	ram
\$9000 - 9FFF	ram	ram	ram	CHARA
\$8000 - 8FFF	ram	ram	ram	ram
\$7000 - 7FFF	ram	ram	ram	ram
\$6000 - 6FFF	ram	ram	ram	ram
\$5000 - 5FFF	ram	ram	ram	ram
\$4000 - 4FFF	ram	ram	ram	ram
\$3000 - 3FFF	ram	ram	ram	ram
\$2000 - 2FFF	ram	ram	ram	ram
\$1000 - 1FFF	ram	ram	ram	CHARA
\$0000 - 0FFF	ram	ram	ram	ram

GAME 1 0 0
EXROM x 0 0
CHAREN x x 0
HIRAM 0 0 0
LORAM 0 0 1

	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 - FFFF	ram	ram	ram	ram
\$E000 - EFFF	ram	ram	ram	ram
\$D000 - DFFF	ram	ram	ram	ram
\$C000 - CFFF	ram	ram	ram	ram
\$B000 - BFFF	ram	ram	ram	ram
\$A000 - AFFF	ram	ram	ram	ram
\$9000 - 9FFF	ram	ram	ram	CHARA
\$8000 - 8FFF	ram	ram	ram	ram
\$7000 - 7FFF	ram	ram	ram	ram
\$6000 - 6FFF	ram	ram	ram	ram
\$5000 - 5FFF	ram	ram	ram	ram
\$4000 - 4FFF	ram	ram	ram	ram
\$3000 - 3FFF	ram	ram	ram	ram
\$2000 - 2FFF	ram	ram	ram	ram
\$1000 - 1FFF	ram	ram	ram	CHARA
\$0000 - 0FFF	ram	ram	ram	ram

GAME 0
EXROM 1
CHAREN x
HIRAM x
LORAM x

	CPU			VIC
	W	R($\overline{BA}$)	R(BA)	R
\$F000 - FFFF	ROM H	ROM H	ROM H	ROM H
\$E000 - EFFF	ROM H	ROM H	ROM H	ram
\$D000 - DFFF	I/O	ram	I/O	ram
\$C000 - CFFF				ram
\$B000 - BFFF				ROM H
\$A000 - AFFF				ram
\$9000 - 9FFF	ROM L	ROM L	ROM L	ram
\$8000 - 8FFF	ROM L	ROM L	ROM L	ram
\$7000 - 7FFF				ROM H
\$6000 - 6FFF				ram
\$5000 - 5FFF				ram
\$4000 - 4FFF				ram
\$3000 - 3FFF				ROM H
\$2000 - 2FFF				ram
\$1000 - 1FFF				ram
\$0000 - 0FFF	ram	ram	ram	ram

Undocumented 6510 Op-Codes (see V6 I05 Page 58) – Raymond Quirling, Kerby, OR

Jim	Op	Mne	Mode	1st	2nd	Comment	Jim	Op	Mne	Mode	1st	2nd	Comment
SAX	cb*	acm	#\$nn	(and)	(cmp)	xr = (ar&xr) - #\$nn:cmp flags	ASO	03	slo	(\$zp,x)	asl	ora	Shift Left into carry: Ora
ASO	0b*	ann	#\$nn	and		Same as 29: and #\$nn But c = 1 if bit 7 = 1	07	slo	\$zp	asl	ora		
RLA	2b*	ann	#\$nn	and		duplicate of 0b	0f	slo	\$addr	asl	ora		
ARR	6b*	arr	#\$nn	and	(ror)	And: Rotate carry Right bit 7 = shift out of c:c = shift out of bit 7 v = shift out of bit 7 or bit 6 z = 1 if result = 0	13	slo	(\$zp),y	asl	ora		
ALR	4b	asr	#\$nn	and	lsr	And, Shift Right into carry	17	slo	\$zp,x	asl	ora		
AXS	83	axs	(\$zp,x)	(and)	store	M = ar&xr:ar,xr,flags unchanged	1b	slo	\$addr,y	asl	ora		
	87	axs	\$zp	(and)	store		1f	slo	\$addr,x	asl	ora		
	8f	axs	\$addr	(and)	store		LSE	43	sre	(\$zp,x)	lsr	eor	Shift Right into carry: Eor
	93*	axs	(\$zp),y	(and)	store	M = ar&xr&\$zz	47	sre	\$zp	lsr	eor		
	97	axs	\$zp,y	(and)	store		4f	sre	\$addr	lsr	eor		
MKA	9f*	axs	\$addr,y	(and)	store	M = ar&xr&\$zz	53	sre	(\$zp),y	lsr	eor		
DCM	c3	dcm	(\$zp,x)	dec	cmp	cmp flags	57	sre	\$zp,x	lsr	eor		
	c7	dcm	\$zp	dec	cmp		5b	sre	\$addr,y	lsr	eor		
	cf	dcm	\$addr	dec	cmp		5f	sre	\$addr,x	lsr	eor		
	d3	dcm	(\$zp),y	dec	cmp		MKX	9e*	xas	\$addr,y	(and)	store	M = xr&\$zz
	d7	dcm	\$zp,y	dec	cmp			9c	yas	\$addr,x	(and)	store	M = yr&\$zz
	db	dcm	\$addr,y	dec	cmp		CIM	02	dth	!!!			death-lockup
	df	dcm	\$addr,x	dec	cmp		12	dth	!!!				death-lockup
INS	e3	isb	(\$zp,x)	inc	sbc		22	dth	!!!				death-lockup
	e7	isb	\$zp	inc	sbc		32	dth	!!!				death-lockup
	ef	isb	\$addr	inc	sbc		42	dth	!!!				death-lockup
	f3	isb	(\$zp),y	inc	sbc		52	dth	!!!				death-lockup
	f7	isb	\$zp,x	inc	sbc		62	dth	!!!				death-lockup
	fb	isb	\$addr,y	inc	sbc		72	dth	!!!				death-lockup
	ff	isb	\$addr,x	inc	sbc		92	dth	!!!				death-lockup
	bb	lsa	\$addr,y	(txs)	store	sp = xr = ar = M&\$F6:AND flags	b2	dth	!!!				death-lockup
XAA	9b*	lss	\$addr,y	(txs)	store	sp = ar&xr: M = ar&xr&\$zz	d2	dth	!!!				death-lockup
LAX	a3	ltx	(\$zp,x)	lda	tax	xr = ar = M	f2	dth	!!!				death-lockup
	a7	ltx	\$zp	lda	tax		NOP	1a	nop				pc + 1
	af	ltx	\$addr	lda	tax		3a	nop					pc + 1
	b3	ltx	(\$zp),y	lda	tax		5a	nop					pc + 1
	b7	ltx	\$zp,y	lda	tax		7a	nop					pc + 1
	bf	ltx	\$addr,y	lda	tax		da	nop					pc + 1
XAA	8b*	oxa	#\$nn	(ora)	(and)	ar = ar!\$EE&xr&#\$nn	fa	nop					pc + 1
OAL	ab	oxx	#\$nn	(ora)	(and)	(tax) xr = ar = ar!\$EE&xr&#\$nn	SKB	04	skb	#\$nn			pc + 2
RLA	23	rln	(\$zp,x)	rol	and	Rotate Left thru carry: aNd	14	skb	#\$nn				pc + 2
	27	rln	\$zp	rol	and		34	skb	#\$nn				pc + 2
	2f	rln	\$addr	rol	and		44	skb	#\$nn				pc + 2
	33	rln	(\$zp),y	rol	and		54	skb	#\$nn				pc + 2
	37	rln	\$zp,x	rol	and		64	skb	#\$nn				pc + 2
	3b	rln	\$addr,y	rol	and		74	skb	#\$nn				pc + 2
	3f	rln	\$addr,x	rol	and		80	skb	#\$nn				pc + 2
RRA	63	rrd	(\$zp,x)	ror	adc	Rotate Right thru carry: aDc	82	skb	#\$nn				pc + 2
	67	rrd	\$zp	ror	adc		89*	skb	#\$nn				pc + 2
	6f	rrd	\$addr	ror	adc		c2	skb	#\$nn				pc + 2
	73	rrd	(\$zp),y	ror	adc		d4	skb	#\$nn				pc + 2
	77	rrd	\$zp,x	ror	adc		e2	skb	#\$nn				pc + 2
	7b	rrd	\$addr,y	ror	adc		f4	skb	#\$nn				pc + 2
	7f	rrd	\$addr,x	ror	adc		SKW	0c	skw	\$addr			pc + 3
	eb	sbc	#\$nn	sbc		duplicate of standard e9 sbc #\$nn	1c	skw	\$addr				pc + 3
							3c	skw	\$addr				pc + 3
							5c	skw	\$addr				pc + 3
							7c	skw	\$addr				pc + 3
							dc	skw	\$addr				pc + 3
							fc	skw	\$addr				pc + 3

Undocumented 6500 Series Instructions (see V6 I05 Page 59) – Noel Nyman, Seattle, WA

Name/Operation	Mode	M.L. Form	Op Bytes	Name/Operation	Mode	M.L. Form	Op Bytes
AAX A&X to M AND A with X	Zero Page Zero Page,Y Pre-Indexed,X Post-Indexed,Y Absolute	AAX \$aa AAX \$aa,Y AAX (\$aa,X) AAX (\$aa),Y AAX \$aaaa	87 2 97 2 83 2 93 2 8F 2	TEA (A&X&(>M)+1)) to M Increment the high byte of the address of M, AND with A AND X	Absolute,Y TEA \$aaaa,Y	9F	3
DCP A-(DECM) Decrement M, Compare with A	Zero Page Zero Page,X Pre-Indexed,X Post-Indexed,Y Absolute Absolute,X Absolute,Y	DCP \$aa DCP \$aa,X DCP (\$aa,X) DCP (\$aa),Y DCP \$aaaa DCP \$aaaa,X DCP \$aaaa,Y	C7 2 D7 2 C3 2 D3 2 CF 3 DF 3 DB 3	TEX (X&(>M)+1)) to M Increment the high byte of the address of M, AND with X	Absolute,Y TEX \$aaaa,Y	9E	3
ISB A-(INCM)-C to A,C Increment M, Subtract from A	Zero Page Zero Page,X Pre-Indexed,X Post-Indexed,Y Absolute Absolute,X Absolute,Y	ISB \$aa ISB \$aa,X ISB (\$aa,X) ISB (\$aa),Y ISB \$aaaa ISB \$aaaa,X ISB \$aaaa,Y	E7 2 F7 2 E3 2 F3 2 EF 3 FF 3 FB 3	AXM (A&X)-M to X AND A with X, subtract M, store in X	Immediate AXM Imm	CB	2
LAX M to A,M to X Load A and X	Zero Page Zero Page,Y Pre-Indexed,X Post-Indexed,Y Absolute Absolute,Y	LAX \$aa LAX \$aa,Y LAX (\$aa,X) LAX (\$aa),Y LAX \$aaaa LAX \$aaaa,Y	A7 2 B7 2 A3 2 B3 2 AF 3 BF 3	RAM LSR(A&M) to A AND A with M, Shift right and store in A	Immediate RAM Imm	4B	2
LAN (ROLM)&A to A Rotate M left, AND with A	Zero Page Zero Page,X Pre-Indexed,X Post-Indexed,Y Absolute Absolute,X Absolute,Y Immediate	LAN \$aa LAN \$aa LAN (\$aa,X) LAN (\$aa),Y LAN \$aaaa LAN \$aaaa,X LAN \$aaaa,Y LAN Imm	27 2 37 2 23 2 33 2 2F 3 3F 3 3B 3 2B 2	RBM ROR(A&M) to A AND A with M, Rotate right and store in A	Immediate RBM Imm	6B	2
RAD (RORM)+A+C to A,C Rotate M right, Add with Carry	Zero Page Zero Page,X Pre-Indexed,X Post-Indexed,Y Absolute Absolute,X Absolute,Y	RAD \$aa RAD \$aa,X RAD (\$aa,X) RAD (\$aa),Y RAD \$aaaa RAD \$aaaa,X RAD \$aaaa,Y	67 2 77 2 63 2 73 2 6F 3 7F 3 7B 3	XMA AND X with M,(X&M)&(AOR#\$EE) to A OR A with #\$EE, AND the two resulting bytes and store in A	Immediate XMA Imm	8B	2
LOR (ASLM)OR A to A Shift M left, OR with A	Zero Page Zero Page,X Pre-Indexed,X Post-Indexed,Y Absolute Absolute,X Absolute,Y Immediate	LOR \$aa LOR \$aa,X LOR (\$aa,X) LOR (\$aa),Y LOR \$aaaa LOR \$aaaa,X LOR \$aaaa,Y LOR Imm	07 2 17 2 03 2 13 2 0F 3 1F 3 1B 3 0B 2	TEY (Y&#\$01) to M test bit 0 in Y	Absolute,X TEY \$aaaa,X	9C	3
REO (LSRM)EOR A to A Shift M right, Exclusive OR with A	Zero Page Zero Page,X Pre-Indexed,X Post-Indexed,Y Absolute Absolute,X Absolute,Y	REO \$aa REO \$aa,X REO (\$aa,X) REO (\$aa),Y REO \$aaaa REO \$aaaa,X REO \$aaaa,Y	47 2 57 2 43 2 53 2 4F 3 5F 3 5B 3	AMA (M&#\$EE)OR(A&M&#\$11) to A	Immediate AMA Imm	AB	2
				LAS M to A, M to X, M to SP Store M in A, X, and SP	Absolute,Y LAS \$aaaa,Y	BB	3
				AXS (A&X) to SP, (A&X&#\$01) to M AND A with X, store in SP, AND SP with #\$01, store in M	Absolute,Y AXS \$aaaa,Y	9B	3
				NPA to NPF Identical with NOP	Implied Implied Implied Implied Implied Implied	NPA NPB NPC NPD NPE NPF	1A 1 3A 1 5A 1 7A 1 DA 1 FA 1
				MNA to MNN Add #\$02 to PC	Implied Implied Implied Implied Implied Implied Implied Implied Implied Implied	MNA MNB MNC MND MNE MNF MNG MNH MNI MNJ MNK MNL MNM MNN	04 2 14 2 34 2 44 2 54 2 64 2 74 2 D4 2 F4 2 80 2 82 2 89 2 C2 2 E2 2
				DNA to DNG Add #\$03 to PC	Implied Implied Implied Implied Implied Implied Implied Implied	DNA DNB DNC DND DNE DNF DNG	0C 3 1C 3 3C 3 5C 3 7C 3 DC 3 FC 3
				SBC Executes identically with the documented SBC op-code \$E9	Immediate SBC Imm	EB	2
				CRA to CRL Processor halts, must be hardware reset	CRA 02 CRB 12 CRC 22 CRD 32	CRE 42 CRF 52 CRG 62 CRH 72	CRI 92 CRJ B2 CRK D2 CRL F2

6510 Opcodes

Raymond Quirling
Kerby, OR

Editor's Note: *These next two articles came in the same batch of mail. They're both about further investigation of the undocumented op-codes. They both have an intersection set of data, but each had other additional observations and examples that we didn't want to exclude. Besides, the two tables together make a good cross reference, considering there are no official specs.*

People who investigate undocumented opcodes deserve all the headaches they get! I started with Joel Shepherd's list in Oct 83 Compute! and discovered a few weird things. Now Jim McLaughlin's list in Transactor Vol 6, Issue 3 has added a few more surprises. I thought I had done everything right but McLaughlin proved me wrong. But after I looked over the list again I still come up with a few disagreements with Jim's list. If he and I are both right it could mean that different 6510's have different responses to the undocumented codes.

The attached listing (page 56) is marked with asterisks where I still disagree with Jim. The first column is Jim's mnemonics (as he listed the code). The 1st operation is performed on the address contents, then the second operation is performed on the new contents. Parenthesis indicate that something non standard is going on.

CB (acm) produces CMP flags: v (overflow) is not affected. 0B (ann) and 2B (ann) are identical and very similar to the normal AND opcode. The c flag is weird and I did not detect any shifting of the immediate location. The clock cycles Jim found also point to a simple AND function. 6B (arr) has some really weird flags! It is not safe to make assumptions about how the flags work. 8B (oxa) and AB (oxx) are both affected by the strange ORA #EE. 89 (skb) is a two byte nop for me, but it does look out of place.

BB (lsa) and 9B (lss) both affect the stack. Neither McLaughlin nor Shepherd indicated this, but Jim did note his difficulty in timing BB (probably impossible to do with the stack flitting about). But the most weird of all is the \$zz store operation that affects five opcodes 93 (axs), 9B (lss), 9C (yas), 9E (xas) and 9F (axs). Shepherd noted the effect on 9F and 9E but was unable to determine the source of the operand for the mysterious AND operation. I figured it out after noting that my operand was different than Joel's and changed after switching to a different memory location to do the testing. After McLaughlin failed to note the effect I repeated it and discovered the surprising result when a page boundary is crossed.

I counted 105 undocumented 6510 opcodes from pages 256 through 259 of the "Commodore 64 Programmer's Reference Guide". They were tested one by one using Butterfield's Supermon. Flag changes were pure drudgery to document. Sorry, no 6502 comparisons. I retained the three character mnemonic description even though it isn't always informative with some of the more involved operations. I liked Jim's skip mnemonics but prefer dth (death) to his CIM (crash immediate). Pick your own personal mnemonics to taste!

The "death" opcodes are suspicious. What trickyness lies hidden therein? Are they interrupts or subroutine calls to an infinite loop? Maybe the kernal is dropped out? I looked for changes to memory, including the stack area, without success.

Are these codes safe to use? That is, can you write software using the new codes for use in other C-64's or with the new C-128? Some of the codes appear to be "accidental" rather than planned. These might change when going from one generation chip to another, or even from one 6510 to another. All the NOP's, SKW's and SKB's fit this category. Be safe! check it out!

I have installed the new opcodes in a disassembler written in Basic. (No assembler yet.) It is available to anyone who sends a disk with return postage to R. Quirling, P.O. Box 135, Kerby, OR 97531.

Legend

ar,xr,yr	register contents	M	contents of address
sp	stack pointer	!	logical OR
c,z,n,v	status flag bits	&	logical AND
\$zp	zero page address	=	equals, or 'is assigned to'
#\$nn	immediate value	*	indicates disagreement
\$addr	absolute address		with McLaughlin findings

\$zz explanation

Used by 93 axs (\$zp),y
9b lss \$addr,y
9c yas \$addr,x
9e xas \$addr,y
9f axs \$addr,y

$zz = \text{msb}(\text{addr}) + 1$

memory = address mode: M = \$zz&(opcode specific value)

If a page is crossed then (surprise):

memory = M * \$100 + lsb(address mode)

Example using 9b lss \$addr,y:

ar	xr	sp	addr	yr	addr,y	memory	zz	M	
3f	ff	3f	500f	00	500f	500f	51	11	(ar&xr&\$zz)
3f	ff	3f	500f	f0	50ff	50ff	51	11	
3f	ff	3f	5010	f0	5100	1100	51	11	
3f	ff	3f	500f	f1	5100	1100	51	11	
3f	ff	3f	5011	f0	5101	1101	51	11	

Example using 93 axs (\$fc),y: 00fc=0f, 00fd=50:

ar	xr	sp	addr	yr	(\$zp),y	memory	zz	M	
ff	3f	xx	500f	f0	500f	50ff	51	11	(ar&xr&\$zz)
ff	3f	xx	500f	f1	5100	1100	51	11	

Example using 9c yas \$500f,\$f1:

ar	xr	sp	addr	yr	addr,x	memory	zz	M	
ff	f1	xx	500f	3f	5100	1100	51	11	(yr&\$zz)

The Undocumented 6500 Series Instructions: A Summary

Noel Nyman
Seattle, WA

Editors Note: *To make this article complete, I felt it best to print Mr. Nyman's opening letter enclosed with the article, to act as a qualifier of the data presented today. After reading his letter, I am sure you will feel that Mr. Nyman is quite adequately qualified to reach the conclusions he has. As a note from us, Noel, thanks for thinking of us. Noel's table has been reproduced on page 57*

Letter by Noel Nyman

I was very interested in "Hidden Op-Codes" by Jim McLaughlin in Transactor 6-03.

Several months ago I and other local programmers worked on these undocumented op-codes for Reflexive Software. The block editor in their Disk Maintenance package will optionally disassemble these op-codes.

While doing the research, we tried most of the codes on several different 65XX series MPU's, including those in PET's, VIC-20's, C64's, Plus4, and an APPLE II. We tried to find 65XX's made by Synertek or Rockwell, the two licensees of MOS Technologies, without success.

All of the op-codes we tested worked identically on all the MPU's we tried.

Mr. McLaughlin's four 'peculiar' codes worked reliably for us on different MPU's, although they gave us different results than his. Readers can easily test the codes by using any ML monitor, storing the short programs on the following pages into memory using the 'M' command, then executing the code with either 'Go' or 'Walk' and examining the registers and memory locations afterwards.

Mr. McLaughlin's chart lists two op-codes which he did not discuss, \$9E and \$9F. He lists as a reference the Joel Shepherd article "Extra Instructions" in the October 1983 COMPUTE! This was the first source we found that gives a function for these two codes.

Shepherd states that the two codes AND registers in the 65XX with \$04, although he didn't know where the \$04 came from.

The Complete Commodore Inner Space Anthology lists these op-codes with the same explanation.

The code will execute much as Shepherd describes, so long as the absolute memory location is in \$03 page. I suspect that Shepherd and his predecessors have used the tape buffer for their test code. The \$9E and \$9F instructions actually use absolute indexed Y addressing and AND the respective registers with the high byte of the memory ADDRESS plus one.

For example, place the test code in the tape buffer area, \$033C and use memory address \$0350. The results are as Shepherd and the Anthology report, \$04 will appear at location \$0351 (Y is loaded with \$01 for the test). Now change the memory address to \$7050 and execute the code. A \$71 appears at location \$7051.

The whole subject of undocumented op-codes is probably at best a curiosity. I've heard that some commercial programs use these codes for security, both West and Butterfield have said so. But I've yet to see an example. I'd be very interested in any differences from our list your readers encounter.

I'm including a copy of the chart we made for Reflexive Software which we believe to be accurate. For your convenience, I've included a disk with the chart, this letter, and the attached pages in EasyScript files.

Sincerely, Noel Nyman, Seattle WA

Some example using Noel's notation follow on the next page.

A Few Examples

\$8B (X&M)&(AV#\$EE) to A (Immediate)

AND X with M, AND the result with (A OR \$EE) and store in A

Sample code:

```
CLC
LDA #$00
LDX #$FF
XAA #$FF
BRK
```

From memory mode, store:

```
. 18 A9 00 A2 FF 8B FF 00
```

After executing, examine the A register.

\$9B (A&X) to SP (A&X&#\$01) to M (Absolute,Y)

AND A with X and store in Stack Pointer. AND Stack Pointer with #01 and store in M.

Sample code:

```
CLC
LDA #$0F
LDA #$07
LDY #$01
XAA $00FB,Y
BRK
```

From memory mode store:

```
. 18 A9 0F A2 07 A0 01 9B
. FB 00 00
```

Before executing code, use the 'Register' mode of the monitor to change the Stack Pointer to \$FF and location \$00FC to \$00. After execution, check the Stack pointer and \$00FC.

\$BB M to A, M to X, M to SP (Absolute,Y)

Store M in A, X, and SP

Sample code:

```
LDX #$00
LDA #$0F
STA $FC
TXA
LDY #$01
LAS $00FB,Y
BRK
```

From memory mode, store:

```
. A2 00 A9 0F 85 FC 8A A0
. 01 BB FB 00 00
```

Before executing, use the 'Register' mode of the monitor to change the Stack Pointer to \$FF. After execution, check A, X and the Stack Pointer.

\$EB A-M to A (Immediate)

Subtract M from A and store in A. This executes identically with the documented SBC code \$E9

Sample code:

```
SEC
LDA #$80
SBD #$01
BRK
```

From memory mode store:

```
. 38 A9 80 EB 01 00
```

After executing code, examine A.

\$9F (A&X&(>M)+1) to M (Absolute,Y)

Increment the high byte of the address of M, AND with A AND X, store in M.

Sample code:

```
LDA #$FF
LDX #$FF
LDY #$01
MKA $7050,Y
BRK
```

From memory mode, store:

```
. A9 FF A2 FF A0 01 9F 50
. 70 00
```

After executing, examine memory location \$7051. If you change the \$70 to \$30 you'll duplicate the code as described by Shepherd.

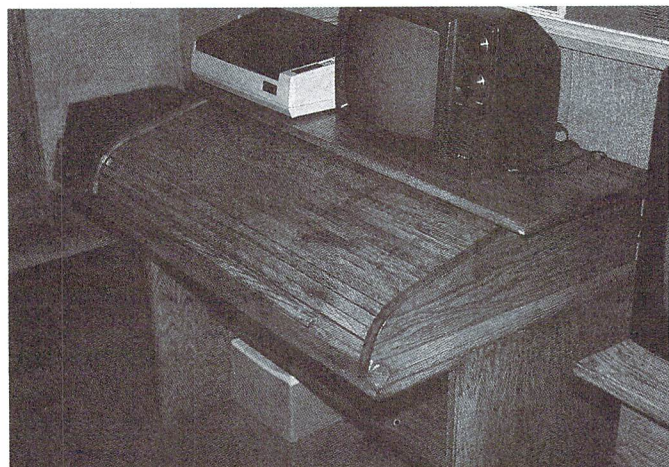
\$9E (X&(>M)+1) to M (Absolute,Y)

Increment the high byte of the address of M, AND with X, store in M.

To test, use the sample code for \$9F and change the op-code to \$9E.

A Computer Rolltop Stand

Mark Jordan
Ligonier, IN



...Not only order, but the *appearance*
of order even if there is none.

Until I built this rolltop computer stand, I was like most of the other computer people I know — disorganized. I had my black and white TV “monitor” mounted on a couple of thick books, sitting behind my 64, my disk drive to the left. Books, magazines, disks and other hi-tech refuse were strewn about. This provision worked until I brought home a printer. I shoved my disk drive over thisaway, moved my TV over thataway, and dropped my printer down on the only bare acreage I could find. A complex array of cords and wires were then tied with little twistee things and carefully positioned behind the desk where everything was sitting to allow space for my paper to snake through. I sat down to compute.

It was too crowded. Reference books were on the floor, opened. Disks were propped between TV and disk drive where magnetic fields are strongest. Demonstrating the printer to my technologically-unsophisticated wife was embarrassing: the paper kept tearing en route to the printer. This put me in a snapping bad mood.

I decided I must build a desktop stand to put some order in my life. The rolltop model depicted here is the result. The stand itself does wonders for my desktop organization but what really makes this piece a winner is the rolltop design. I now am a zealot proclaiming the virtues of having a two-tiered, rolltop computer workplace.

You don't have to be a carpenter or fine woodworker to build one. Basically the project consists of five parts. Only two are tricky: routing a groove for the tambour (the sliding part) to slide in and building the tambour itself. And even these two aren't *too* tricky.

The first order of business is to buy your wood. The only problem is, I can't tell you the exact dimensions because you are going to customize this stand to fit your workspace. But you can easily figure what lumber you'll need by performing the following simple calculations. Please refer to the exploded drawing (figure 1) throughout.

First, to simplify matters and to save you time and money, all the lumber in the project will be common stock. The sides will be two by eights, the brace a two by four, the top a twelve inch wide shelf piece, and the slats three-eighths by three-quarter inch moldings.

To figure the length of the brace (C), simply subtract four inches from the top's length.

Finally, to determine the number and length of slats you'll need for the tambour, perform these calculations. Take half of the sides' **length** and add two inches. Now divide this number by .75. The result equals the number of slats you'll need since each slat will be three-quarters inch wide. The **length** of the slats will simply be the length of the brace plus three-fourths inch.

You may have to do some asking to find out just what is available for your slats. Be sure to ask about window and door moldings and lattice. If your yard doesn't carry anything that's three-eighths inch thick by three-quarters inch wide, you can simply readjust your calculations for how many pieces you'll need and buy whatever widths they have. don't buy any pieces wider than one inch because you'll have difficulty with them negotiating the curve when they are sliding. Feel free to use as narrow of stock as they sell.

The next order of business is to buy (or scrounge) a piece of fabric to glue your tambour to. It needs to be tough material (denim is good) and it needs to be as big as your finished tambour will be.

You are now ready to start building. Cut all pieces to the proper length. Now refer to figure 1 and notice that the sides begin sloping on the top edge midway along its length (see point 1). Mark that spot on your piece. Next note point 2 on figure 1. It is located three inches up from the bottom of the front of the side piece. Mark this on your sides and then use a straightedge to connect points 1 and 2 and cut them along this line.

The curve is not official science. If you have good sanding equipment (like a belt or disk sander), then you might just sand a gentle curve. Lacking one, you could nibble away at the sharp corner first with a saw until you get it to the hand-sandable stage. Anyway, you need to end up with a nice flowing curve. And, you need to make both sides identical. So sand the finishing touches with the two pieces clamped together.

The tambour groove needs to be done next. You need a router unless you like chiseling. If you have a router with an adjustable guide, use it to make the groove. Lacking a good guide, make a template for the router to ride against and clamp it to the side piece while routing.

A word of caution here. Routing is to woodworking what the GO and RUN commands are to programming. Things happen fast and sometimes, stuff gets messed up. That's why smart programmers save their work before a Go or RUN and that's why smart woodworkers only take a little wood at a time when they rout. In fact, they first do a dry run, checking to see exactly where the bit will byte (sorry, I'm getting as sick of those puns as you are but you have to admit, it really fit there).

The depth and width of the groove are recorded in figure 1 but I'll repeat them here: one-half inch by one-half inch. This will allow for just enough slop in your tambour for it to slide easily.

Now, on to the top. Cut it to size and it's done.

Now, on to the brace. Cut it to size and it's done.

Now, on to the tambour. Cut all the slats to size and lay them together tightly on a large sheet of scrap plywood or anything flat. Next tack down a couple of scrap wood pieces to keep them tight and square. Get some Elmer's Glue-All or other wood/fabric glue, pour it all over the slats, rub it around with your fingers until the slats are well covered, and take your piece of fabric and lay it onto this gooey surface. Press it down until you are sure the fabric has made full contact. Keep pressing until the fabric stays put (about fifteen minutes).

Let the tambour dry for a day.

Assembly time. Stand the sides on edge and set the top on it. Position the sides so that you allow a one-inch overhang of the top to the sides. Square everything up, then carefully remove the top and apply glue to the top edge of the side pieces. Carefully replace the top, re-square everything and either screw or nail the top in place. For looks, if you screw it down, be sure to countersink the screws so that you can cover them later with dowels or putty.

Now lay a bead of Elmer's on the top edge of the back brace and slip it up between the sides and under the top. It will be flush with the back of the top and sides (see figure 2). Nail or screw it from the top.

Finally, take your tambour and guide it up into the open groove from the bottom of the sides at the front and slide it all the way up and back. You will find it extends out about an inch from the top piece. If you wish, you can fashion a handle to pull the tambour with (see finished drawing), or you can just pull on the tambour itself. A handle is nothing more than a piece of molding glued to the bottom slat in the tambour.

Except for the finished work (sanding, staining, varnishing, etc.) you are done. Before doing any of that dreary stuff, hustle your new computer addition into the house and set it on your desk. Put everything the way you want it. Instant organization.

Now for the clincher. Pull down the tambour and, Eureka! everything is hidden. Your desk looks neat. All disks, books, Coke cans—everything—is hidden beneath that wonderful sliding front.

Well, it just goes to show you, there are some bargains left in life. You built this stand to organize your work station, and you got not only order, but also, the *appearance* of order even when there is none. And you got it all for only a few hours labor. Now that's what I call a good deal.

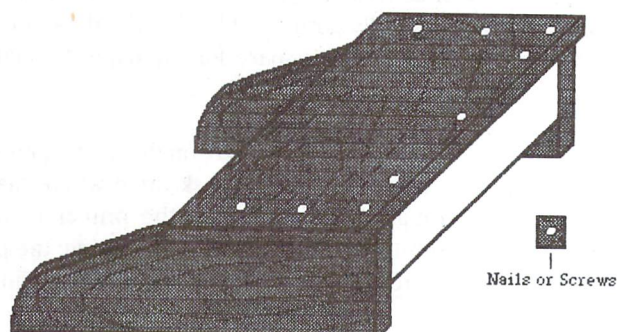
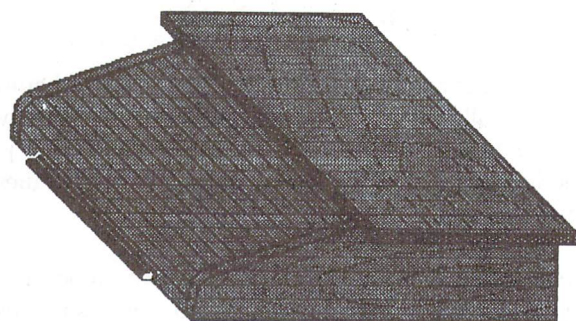
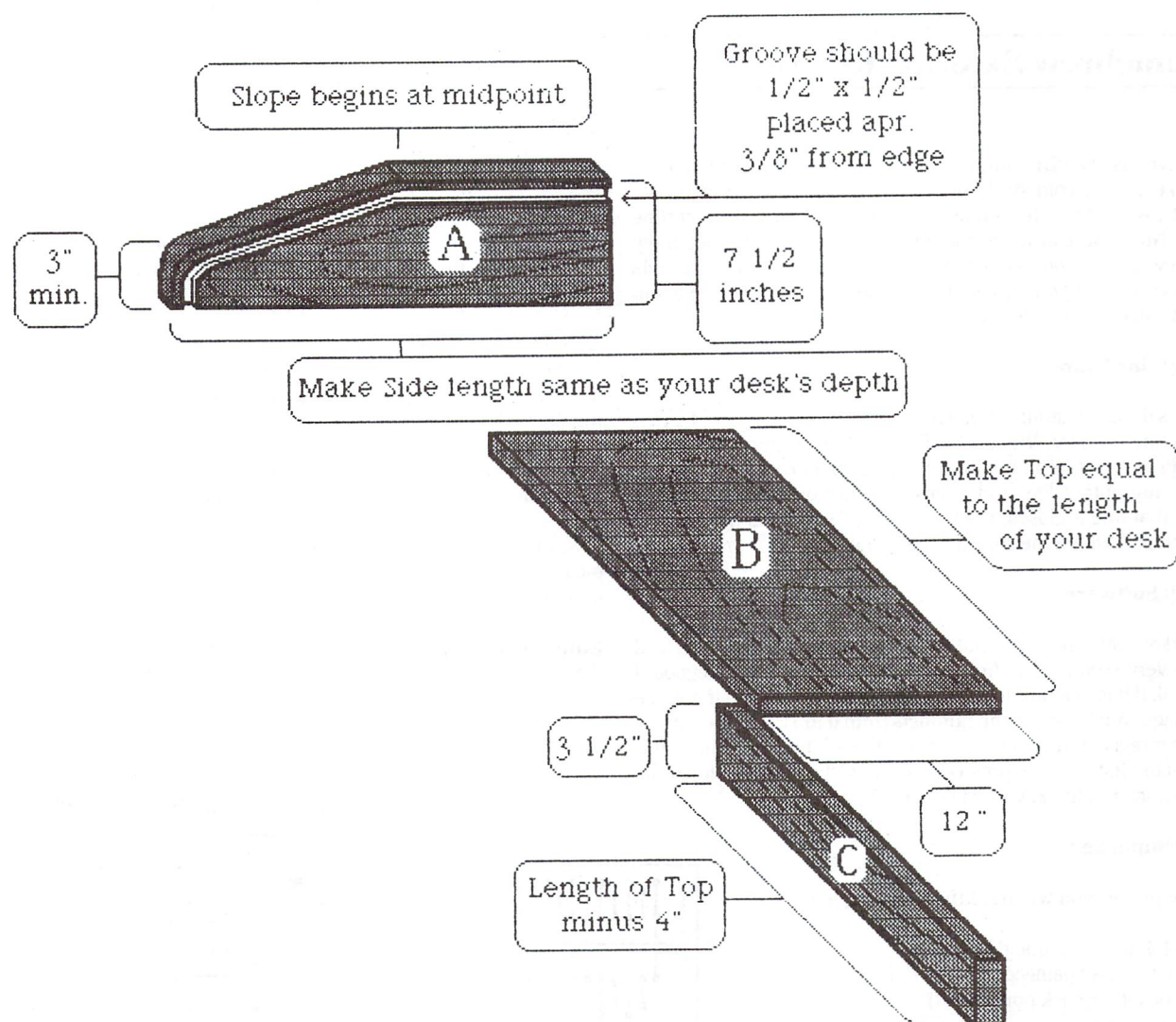


Figure 2



The Finished Rolltop



Exploded View of Rolltop Assembly

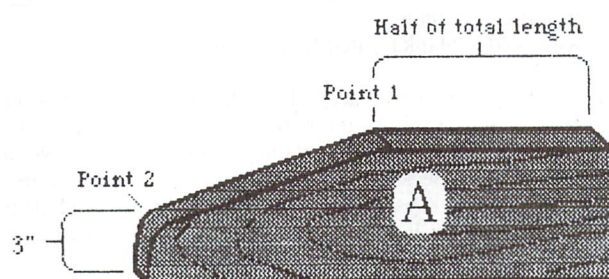


figure 1

Superkey

Andrew Walduck
Orillia, Ontario

A Homebrew External Keypad

If you've ever used a PET computer you know how nice it is to have a numeric keypad at your beck and call. You can blitz through data statements, whiz through number crunching and blast through line numbers. Now you can have this for your 64 and more. Superkey adds 16 new keys to your 64 that repeat, are redefinable and available for your use in either program or direct modes, and the best part about it is that it will cost you under \$30.

Superkey Hardware

Superkey is designed around 1 major integrated circuit, a 74154 TTL 4 line to 16 line decoder. What a 74154 does is take the 4-bit binary number on its inputs (coming from the joystick port) and makes the corresponding output low and keeps all other outputs high. If the switch on that line is closed, the low signal is applied to the input of the joystick port where the software can tell if that key is pressed.

Superkey Software

The Superkey software is tied into the IRQ vector so that it is operated 60 times every second. The first thing it does is see if the keyboard buffer is full. If it is, it jumps to the normal interrupt routine. If not, the software sequentially writes the numbers from 0 to 15 into the second joystick port register, reading the status of the 4th bit each time. If it is low, it means that a key is pressed and the software then stores the correct character in the keyboard buffer.

Building Superkey

To build Superkey you will need the following parts

- 1 74154 4 line to 16 line decoder
- 1 16-key keypad (jameco part# k-19 is good)
- 1 d-9 socket (joystick port socket)
- 1 24 pin ic socket
- miscellaneous items
- wire, solder, blank printed circuit board

After constructing Superkey according to the schematic, double check the wiring to make sure that it is correct and then plug it into joystick port 2 and turn the computer on. If the computer doesn't power up in the usual fashion, check your wiring again. If all is ok so far, press and release all of the 16 keys individually. None of them should have any effect on the screen. If any do, once again check the wiring of the circuit.

Type in and run the following program (don't forget to save it). If no error message is printed, make sure the keypad is plugged into joystick port 2 and enter:

SYS 49152

to enable Superkey. Now pressing the keys on the keypad should have an effect on the screen (ie "0" on the keypad should print 0 on the screen). If something appears wrong, recheck the program and the wiring of the keypad. To disable Superkey enter:

SYS 49155

Suggestions

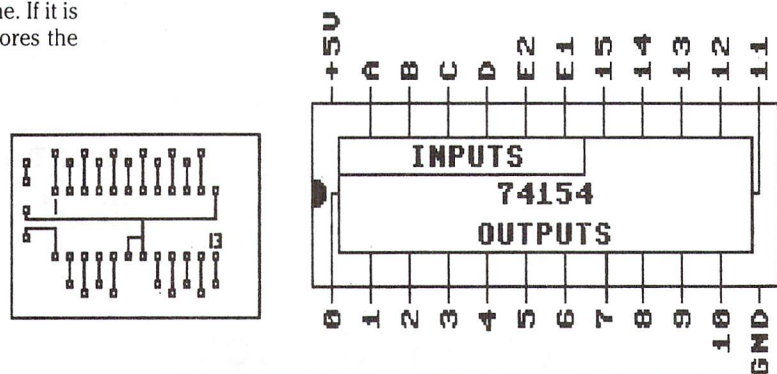
To change the keycodes that Superkey prints, change the data statements beside the rem statements that tell you the key number, to the ASC() value of the character you would like printed when that key is pressed.

One setup that I use quite frequently for entering data statements is a keypad with the numbers 0 to 9, a return key, a "d" key, a shifted a", the delete key, a comma key, and the shift-return key. To do this merely, redefine the various keys on the keypad.

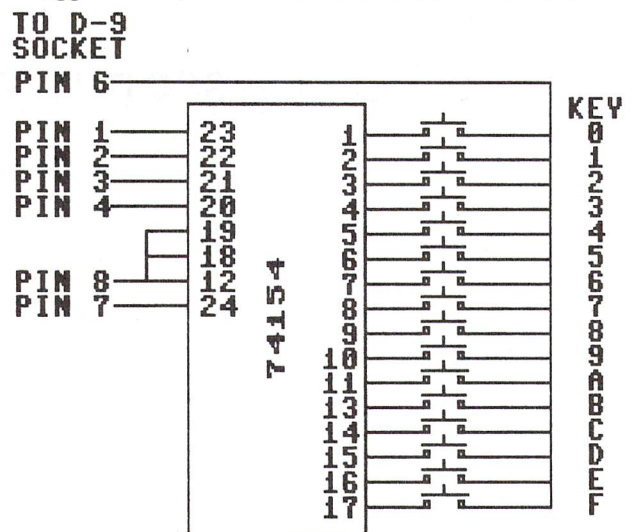
Another setup that I use regularly is one with the cursor keys set up in a diamond pattern with the home key in the center.

If you have any suggestions, problems or ideas regarding Superkey please write me. Andrew Walduck, 441 Barrie Rd #52, Orillia, Ontario, L3V 6T9

Editor's Note: You may not end up building Andrew's keypad, but the program shows how you would service the joystick port in machine language should you ever need to.



Suggested Foil Side of PCB and Pinout of 74154 IC



Schematic for Superkey 16-key Keypad

Superkey Basic Loader

GI	10 rem* superkey *	LK	1160 rem* key definitions follow *
CJ	20 rem* software driver for ext. keypad	OK	1200 data 48 :rem key #0
CJ	30 :	PL	1210 data 49 :rem key #1
BA	40 fori = 49152to49266:readda:ck = ck + da:pokei,da:nexti:fori = 49267to49281:readda	LK	1220 data 50 :rem key #2
HN	50 pokei,da:nexti:ifck<>14190thenprint " error in data statements ":end	ML	1230 data 51 :rem key #3
IG	60 print " superkey now in memory \$c000 to \$c081 ":print " sys 49152 to activate "	NM	1240 data 52 :rem key #4
FH	70 print " sys 49155 to deactivate ":end	ON	1250 data 53 :rem key #5
EM	80 :	PO	1260 data 54 :rem key #6
FO	1000 data 76, 18, 192, 120, 173, 112, 192, 141	AA	1270 data 55 :rem key #7
FN	1010 data 20, 3, 173, 113, 192, 141, 21, 3	BB	1280 data 56 :rem key #8
AP	1020 data 88, 96, 120, 173, 20, 3, 141, 112	CC	1290 data 57 :rem key #9
FI	1030 data 192, 173, 21, 3, 141, 113, 192, 169	PM	1300 data 65 :rem key #10
KB	1040 data 47, 141, 20, 3, 169, 192, 141, 21	BO	1310 data 66 :rem key #11
OD	1050 data 3, 169, 16, 133, 254, 88, 96, 173	DP	1320 data 67 :rem key #12
LD	1060 data 137, 2, 197, 198, 240, 57, 169, 15	FA	1330 data 68 :rem key #13
CP	1070 data 141, 2, 220, 160, 0, 140, 0, 220	HB	1340 data 69 :rem key #14
HC	1080 data 173, 0, 220, 41, 16, 208, 28, 196	EA	1350 data 70 :rem key #15
GN	1090 data 254, 208, 8, 198, 253, 208, 27, 169		
MF	1100 data 4, 208, 2, 169, 20, 133, 253, 185		
MI	1110 data 114, 192, 166, 198, 157, 119, 2, 230		
JM	1120 data 198, 208, 5, 200, 192, 16, 208, 213		
NO	1130 data 132, 254, 169, 255, 141, 2, 220, 76		
DP	1140 data 49, 234		
CP	1150 :		

Superkey PAL Source Code

EO	100	sys700 ;pal 64 assembler	LI	460	sta	pkey
MP	110	; superkey driver program	EG	470	cli	
JG	120	* = \$c000	MM	480	rts	
KO	130	.opt oo	AG	490	;	
CA	140	;	LD	500	start	lda bufmax
PB	150	bufnum = 198	GG	510	cmp	bufnum ;check if keybuffer full
DC	160	bufmax = 649	NL	520	beq	oldirq ;branch if yes
NN	170	datapt = \$dc00	BL	530	lda	##00001111
GL	180	ddr = \$dc02	PL	540	sta	ddr ;set up data direction register
MD	190	keybuf = 631	IL	550	ldy	#\$00 ;start key count at zero
EF	200	irqvec = 788	LC	560	kread	sty datapt ;store count in port 2
AB	210	cntr = \$fd	BM	570	lda	datapt
HP	220	pkey = \$fe	BC	580	and	##00010000 ;check if key pressed
OG	230	delay = 20	DK	590	bne	nokey ;branch if not pressed
OO	240	rpt = 4	EN	600	cpy	pkey ;is current key same as last
PK	250	lkey = 16	EP	610	bne	not ;branch if not the same
KH	260	;	LA	620	dec	cntr ;decrement count until repeat
KL	270	jmp setup ;connect	IC	630	bne	reset ;branch if not time for repeat
JO	280	sei ;disconnect	PO	640	lda	#rpt
HB	290	lda oldirq + 1	BL	650	bne	store
FN	300	sta irqvec	IB	660	not	lda #delay
PC	310	lda oldirq + 2	PK	670	store	sta cntr ;set up counter for next repeat
LK	320	sta irqvec + 1	KB	680	lda	table,y ;lda with character from table
IN	330	cli	DD	690	ldx	bufnum ;current key in keybuffer
AE	340	rts	NB	700	sta	keybuf,x ;store char
EN	350	;	GK	710	inc	bufnum ;add 1 to # of chars in buffer
IH	360	setup sei	AG	720	bne	end ;branch always!
NN	370	lda irqvec	BL	730	nokey	iny ;add 1 to current key number
PK	380	sta oldirq + 1	EB	740	cpy	#lkey ;is loop doneprint
DL	390	lda irqvec + 1	NM	750	bne	kread ;branch if not done
HM	400	sta oldirq + 2	OA	760	end	sty pkey ;save y in previous key pressed
FL	410	lda #<start	MG	770	reset	lda ##11111111
NE	420	sta irqvec	CO	780	sta	ddr ;restore data direction registr
FM	430	lda #>start	GH	790	oldirq	jmp \$ea31 ;go to old irq routine
DC	440	sta irqvec + 1	CI	800	table	.asc "0123456789abcdef"
DD	450	lda #lkey	MC	810	;	table of characters for keys
			AB	820	.end	

String Calc

Daniel Bingamon
Batavia, Ohio

Here is a simple and yet transportable program written in Basic for evaluating expressions in strings.

The program will evaluate the string in a left to right order (not algebraically) like the way you would enter it on a very simple calculator. The program is shown below with a description of what each section does:

First, a simple screen clear.

```
10 printchr$(147)
```

The user is prompted for a formula, passed from a different subroutine instead of the input statement.

```
20 input "enter formula ";f$
```

Program is terminated if stop is typed.

```
30 iff$ = "stop" thenstop
```

The variable "A" will be used as an accumulator for the result.

```
40 a = 0
```

A space is put on the end of the formula as a terminator in case the user did not supply one. "P\$", Is the work string for the parser and "ID" is flag which determines what type of operation will be performed, "9" is the initial mode. "DN" is the finish flag.

```
50 f$ = f$ + " ":p$ = f$:id = 9:dn = 0
```

This loop looks to see the length of an element by looking ahead for the next separator character (space). If it not found then an error message occurs. An error should not occur on line 70 since line 50 insures against it.

```
60 fori = 1 to len(p$)
65 ifmid$(p$,i,1) = " " then80
70 next:print "?invalid formula ":goto20
```

Now a check to see if the program finished flag should be set by determining if the pointer to the end of formula matches the size of the formula.

```
80 ifi = len(p$) thendn = 1
```

Set up string from the work string to the next separator (space).

```
90 x = i:x$ = left$(p$,x)
```

Set value flag on. The Value flag off means that the next character is a symbol. If it is a value then the amount will be placed in the variable "V".

```
100 va = 1:v = val(x$)
```

If the contents of X\$ is not a value then set the value flag back to off.

```
105 ifval(x$) = 0 and left$(x$,1) <> "0" thenva = 0
```

If the symbol is a PI symbol or the word PI then toss PI into V and set value flag back to on.

```
110 ifleft$(x$,2) = "pi" thenv = 3.14159265:va = 1
120 ifleft$(x$,1) = "π" thenv = 3.14159265:va = 1
```

If value flag is on then goto process value routine otherwise find out what type of symbol is used.

```
130 ifva = 1 then230
```

Handle subtotals.

```
135 ifleft$(x$,1) = "'" thenprint "st = ";a:goto310
```

Load first position into string and compare.

```
140 id$ = left$(x$,1):id = 0
```

Compare to possible symbols and set up a symbol code.

```
150 ifid$ = "+" thenid = 1
160 ifid$ = "-" thenid = 2
170 ifid$ = "*" thenid = 3
180 ifid$ = "/" thenid = 4
190 ifid$ = "↑" thenid = 5
```

If symbol could not be found then spit out error.

```
195 if(id > 0) and (id < 90) and mid$(x$,2,1) <> " " then
print "?error ":goto20
```

Skip over extra spaces.

```
200 ifid$ = " " then310
```

If accumulator is still zero then skip test for rounding.

```
203 ifa = 0 then210
```


Round by tens requested?

```
204 ifid$ = " @" thenam = val(mid$(x$,2,1))
      :am = -am:goto207
```

Round by tenths requested?

```
205 if id$<> "; " then210
```

Round by raising to the specific power to make the affected position lie in the tens position, add .5 to the number, truncate and reverse the process to take the number back to original power.

```
206 am = val(mid$(x$,2,1)):ifa = 0then210
207 a = int(a*(10↑am) + .5):a = a/(10↑am)
      :id = 0:goto310
```

If no ID code for symbol found then it must be an invalid symbol, spew out error message.

```
210 ifid = 0thenprint "?bad symbol ":goto20
```

Symbol processed. Get then next element. Normally it should be a value unless, the accessory functions such as rounding or subtotals are used.

```
220 goto310
```

Take the accumulator and the value being processed and merge them by the instruction code created by the Symbols.

```
230 ifid = 1thena = a + v
240 ifid = 9thena = v:id = 0:goto310
250 ifid = 0thenprint "?error ":got20
260 ifid = 2thena = a - v
270 ifid = 3thena = a * v
280 ifid = 5thena = a ↑ v
```

Division by zero trap.

```
290 ifid = 4andv = 0thenprint "?div by zero ":goto20
300 ifid = 4thena = a/v
```

If done flag set exit here to show result and request new formula.

```
310 ifdn = 1then330
```

Move forward to the next element by removing the old elements as we go along.

```
320 p$ = right$(p$, (len(p$)-len(x$))):goto60
```

Display result and clear accumulator.

```
330 print " result = ";a:a = 0:goto20
```

Sure, it doesn't do everything the internal machine language routine that evaluates expressions in your Commodore computer does but, it helps give you a BASIC idea on how a computer can perform these functions. The program can be used in a number of applications and might even be expanded to give some operations priority and even process the parenthesis () symbols.

String Calc, uninterrupted

```
10 printchr$(147)
20 input " enter formula ";f$
30 if f$ = " stop " thenstop
40 a = 0
50 f$ = f$ + " ":p$ = f$:id = 9:dn = 0
60 fori = 1tolen(p$)
65 ifmid$(p$,i,1) = " " then80
70 next:print "?invalid formula ":goto20
80 ifi = len(p$)thendn = 1
90 x = i:x$ = left$(p$,x)
100 va = 1:v = val(x$)
105 ifval(x$) = 0andleft$(x$,1)<> " 0 " thenva = 0
110 ifleft$(x$,2) = " pi " thenv = 3.14159265:va = 1
120 ifleft$(x$,1) = " π " thenv = 3.14159265:va = 1
130 ifva = 1then230
135 ifleft$(x$,1) = " ' " thenprint " st = ";a:goto310
140 id$ = left$(x$,1):id = 0
150 ifid$ = " + " thenid = 1
160 ifid$ = " - " thenid = 2
170 ifid$ = " * " thenid = 3
180 ifid$ = " / " thenid = 4
190 ifid$ = " ↑ " thenid = 5
195 if(id>0)and(id<90)andmid$(x$,2,1)<> " " then
      print "?error ":goto20
200 ifid$ = " " then310
203 ifa = 0then210
204 ifid$ = " @" thenam = val(mid$(x$,2,1))
      :am = -am:goto207
205 if id$<> "; " then210
206 am = val(mid$(x$,2,1)):ifa = 0then210
207 a = int(a*(10↑am) + .5):a = a/(10↑am)
      :id = 0:goto310
210 ifid = 0thenprint "?bad symbol ":goto20
220 goto310
230 ifid = 1thena = a + v
240 ifid = 9thena = v:id = 0:goto310
250 ifid = 0thenprint "?error ":got20
260 ifid = 2thena = a - v
270 ifid = 3thena = a * v
280 ifid = 5thena = a ↑ v
290 ifid = 4andv = 0thenprint "?div by zero ":goto20
300 ifid = 4thena = a/v
310 ifdn = 1then330
320 p$ = right$(p$, (len(p$)-len(x$))):goto60
330 print " result = ";a:a = 0:goto20
```


Getting On Viewtron: It's This Easy!

First, you need the Viewtron Starter Kit. If you live in the U.S. you can call their toll free number:

Viewtron
1 800 272 5400
1111 Lincoln Road, 7th Floor
Miami Beach, Florida
33139

It's just \$9.95 and it comes with handy 46-page reference guide, a disk with software that works superbly, and an hour of Viewtron free.

Viewtron is fast, flexible, versatile, capable, and, it's in color! With a nice color monitor, some of the screens are rather pleasing. The software is somewhat specialized for the NAPLPS videotext protocol, but it can download, send to printer, redial, plus lots of other stuff – it's all in the manual.

Canadian orders that get stuck in customs can take as long as six weeks to get to you from Miami. Instead, let us send you one – it's about the same price after exchange and you can use our postage paid subscription form. See News BRK for more details.

The simplest part about Viewtron has to be signing on. The disk is 4040/1541 format. You LOAD "*" ,8,8 (or ' ,8,1' if you like) and you'll get two options. 1 to sign on, 2 for set-up. Once you get the Viewtron software set up with your ID and Password, hit 1 for sign-on and the program does the rest. If you don't have an auto dial modem you'll need to call your local network manually, but after that the program does the rest. It pumps out the Viewtron call address for you, followed by your ID and Password. The first screen you see is the main index which informs you of any new mail, current events, and away you go. You're On!

Setting Up For Viewtron

The "set-up" option will rarely be used more than once. Before loading the Viewtron software, however, you should first load another program that comes on the disk:

```
load " phone " ,8,8
run
```

Enter your area code and a list of phone numbers will be displayed for the major telecommunications networks (see below). You can access Viewtron from any of these networks so pick a number (preferably a local call) and write it down along with the network it belongs to. Some numbers may not appear in this list but it's easy to change these parameters should you find ways to fine tune your software later on (ie. more local call or 1200 baud line).

Now load the Viewtron program (load "*" ,8,8) and choose option 2. The menu will show three sections that need to be completed.

1: Modem

The following has been reproduced from the "set-up" option and should give you an idea of which modems are compatible with the Viewtron software.

- 1 Volksmodem 1200
- 2 Westridge 6420
- 3 Hesmodem I
- 4 Smartmodem
- 5 1064 Modem
- 7 Commodore 1660
- 8 More modems, takes you to:

- 1 Vicmodem 1600
- 2 64modem
- 3 Volksmodem 6420
- 4 Commodore 1650
- 5 Commodore 1670
- 6 Other auto (dialing) 1200
- 7 Other auto (dialing) 300
- 8 More modems, takes you to:

- 1 Manual 1200
- 2 Manual 300
- 8 More modems, repeats from start.

2: Built in Networks

- 1 Telenet
- 2 Uninet
- 3 LADT
- 4 Tymnet
- 5 Datapac
- 6 Other

Selecting any of the above will prompt for a phone #. This is the local number for the network you have chosen (which you may have found from the "phone" program earlier). Option 6 allows you to enter the phone number of any network, followed by the call sequence to access, theoretically, any service. However, the Viewtron software is somewhat specialized and compatibility with other online services, NAPLPS or otherwise, has not been tested.

3 ID and Password

Option 3 prompts for your Viewtron ID and Password from the card inside the Viewtron Pack. Enter these as shown but hang

on to the card – the ID and Password are sent automatically by the software when you sign on, but some sections of Viewtron will ask you for your password so keep it handy.

Once back at the menu, hit Return and everything will be stored to disk. Don't worry about mistakes, changes are easy. You can even change your password after you get your Viewtron account established.

Your First Time On Viewtron

Now select option 1: Sign-On. If you have an auto-dial modem, sit back and relax. If not, dial the number for your local network manually and wait for the connect tone. Signing on generally takes about 30 seconds but any more than a minute means try again – hit C= and R for Redial.

You'll be asked for your first name, your last name, address, etc., and get your credit card, you'll need that too.

Once Viewtron has all your particulars, you're ready to wander. You may find it strange to have 'New Mail' waiting for you since this is your first time on, but read it anyway for the experience. Try 'CB' – you'll be asked to enter your "handle" so you may want to think one up ahead of time. Try some of the other keywords listed in the manual. It doesn't take long and Viewtron will as easy as loading the software.

The Transactor is currently investigating the potential for a 'Transactor' section on Viewtron. Articles would be made available as well as programs for downloading. To get an idea of what a Transactor section might be like, try the Protecto (enter *protecto*) section. Entering *transactor* would take you to a similar screen where you would select from a number of options.

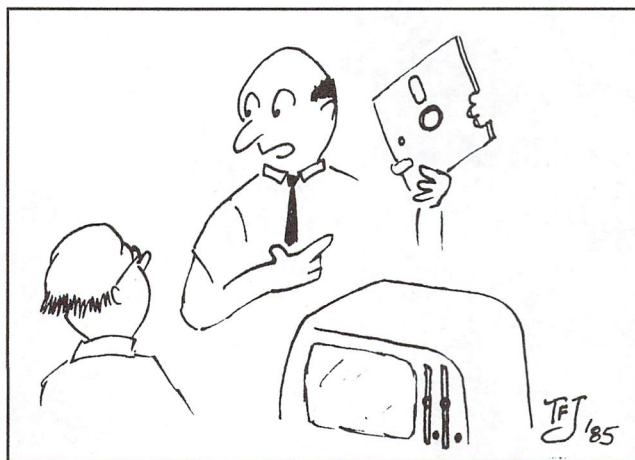
So check it out and let us know what you think. We'll have more about Viewtron next issue.

The SAVE@ Saga

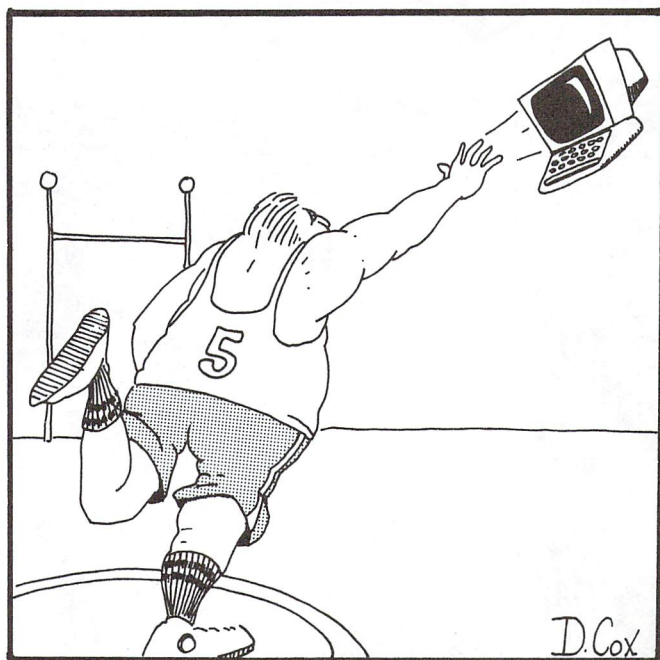
If you've been following the SAVE@ saga you've probably read the brilliant investigative work of P.A.Slaymaker in Compute. Part 2 of Phillip's article uncovered some important missing pieces of the mystery. Unfortunately we didn't have time to make any additional study for this issue. Phil has agreed to make more of his discoveries available to us, including the code changes he discussed in Compute, hopefully by next issue.

The latest news, however, is that SAVE@ is equally offensive in the MSD drives (as noted by Roy Dancy, Dothan, AL) as well as the new Commodore 1571. It's interesting how the same bug can get from one DOS to another, even with two completely different manufacturers. Hmm.

Compu-toons



I told you this new anti-copy protection doesn't fool around.



"INPUT... OUTPUT... SHOT PUT"

CAPTAIN SYNTAX



The Transactor Disk

With only 16 characters per filename, many of the programs on Transactor Disks have somewhat deceiving titles. Most are self-explanatory, others have been appended with additional briefings.

Programs with block counts of only 1 or 2 are usually just subroutines for use within other programs. A "dazzler" is our way of describing the more entertaining (as opposed to useful) programs. And a "demo" indicates illustration of a concept as opposed to a demonstration of the cosmetic variety. A 'V1' in these descriptions indicates Version 1 – look for a similar filename on a later disk for an update. 'ML' denotes

Machine Language, '.bas' is a BASIC file, and '.pal' or '.src' is a PAL Assembler source code file.

TransBASIC files propagate forward to each new disk. Descriptions of TransBASIC files will appear only on the first disk on which they were offered. However, most TB modules can not possibly be described in brief. Commands within modules are listed in REM statements – complete descriptions of the commands can only be obtained from the magazine. Summaries of all commands can be found in the TransBASIC column. The prefix 'TB:' will be used in this list to show TB command files.

"transactor disk1"			
1	"-vol 4 issue 01-"	prg	
1	"opt illusion 4.0"	prg	dazzler
23	"unassembler 4.0"	prg	
4	"string thing 4.0"	prg	4.0 string utility
14	"bmbstringthing"	seq	source code
2	"stringthing.bin"	prg	object code at \$7e00
7	"chain tracer"	prg	traces disk files
2	"ieee modem drivr"	prg	
2	"keyboard setup"	prg	indexed arrays demo
9	"SPet terminal"	prg	in basic
7	"vic aid.rel"	prg	tiny basic aid
3	"vic modem driver"	prg	
1	"-vol 4 issue 02-"	prg	
1	"ftoutsm 4.0"	prg	dazzler
2	"brain bender 4.0"	prg	dazzler
1	"vertical message"	prg	vertical printing demo
1	"vert msg part2"	prg	
2	"screen marker"	prg	cursor positioning
2	"writing a file"	prg	basic code from
2	"reading a file"	prg	an article by
1	"coding comments"	prg	jim butterfield
3	"translator.src"	prg	a translation table
2	"translator.ldr"	prg	utility for basic 4
3	"translator demo"	prg	
12	"sid friend part1"	prg	c64 sid keyboard demo
8	"skiffle band 1"	prg	programmed sid music
7	"skiffle band 2"	prg	
22	"basic aid editor"	prg	extension to basic aid
6	"joystick reader"	prg	
1	"-vol 4 issue 03-"	prg	
3	"kaleidoscope 80"	prg	dazzler
4	"disk append 4.0"	prg	append to basic text
7	"string thing 64"	prg	string utility
2	"tapemaker 64"	prg	send disk files to tape
6	"univ disk change"	prg	change device #
3	"screen center"	prg	center text demo
11	"catstrapolator"	prg	catalog info retrieval
11	"catstrapolator64"	prg	of blocks free, ID, etc.
5	"sid friend part2"	prg	
2	"char sets prog1"	prg	move c64 char set
2	"char sets prog2"	prg	from rom to ram
2	"char sets prog3"	prg	
6	"character editor"	prg	for hi-res chars
10	"high res dumper"	prg	to 8023 printer
15	"labeler.src"	prg	for labelling basic
6	"label loader.vic"	prg	to allow goto & gosub
6	"label loader.c64"	prg	to use labels instead
2	"label test prog"	prg	of line numbers.
12	"c64 tiny aid ldr"	prg	c64 tiny basic aid
1	"-vol 4 issue 04-"	prg	
2	"1 line squiggle"	prg	dazzler
2	"cathode ray prg1"	prg	dazzler
1	"cathode ray prg2"	prg	dazzler
3	"datadjuster"	prg	for controlling data reads V1
2	"retina wrencher"	prg	dazzler
2	"wordpro lister"	prg	
8	"hard disk backup"	prg	9060/9090 to floppies
10	"superkey 64"	prg	keyword burst keys
5	"redecode 8032"	prg	keyboard interception demo
4	"sid friend part3"	prg	
6	"sound help"	prg	sid sound utility
3	"sound help demo"	prg	
23	"sprite palette"	prg	sprite editor
20	"graphics utility"	prg	hi-res utility
31	"graphic util.src"	prg	
6	"raster irq demo"	prg	c64 raster interrupts
13	"raster irq.src"	prg	
1	"-vol 4 issue 05-"	prg	ref. issue - no programs
1	"-vol 4 issue 06-"	prg	
8	"incrementation"	prg	dazzler
2	"moneyout"	prg	formatting dollar figures
5	"palindrome"	prg	see dictionary
2	"autoliner 4.0 v1"	prg	auto line# generate
2	"autoliner 64 v1"	prg	for entering progs V1
4	"datadjust update"	prg	datadjuster V2
15	"refield"	prg	extension for The Manager
23	"create"	prg	for use with 'refield'
2	"updating 4.0"	prg	updating programs demo
9	"mid string dream"	prg	mid\$ demo
1	"get subroutine 1"	prg	
2	"get subroutine 2"	prg	
3	"get subroutine 3"	prg	
8	"menu type 1"	prg	menu driver demos
5	"menu type 2"	prg	
6	"menu type 3"	prg	
2	"directory type 1"	prg	display disk
2	"directory type 2"	prg	dirs from basic
4	"date subtrn 1"	prg	input valid
3	"date subtrn 2"	prg	date strings
3	"st interrogate"	prg	status word handling
8	"univ disk rtns"	prg	basic disk file readers
6	"sid friend part4"	prg	
38	"sid friend part5"	prg	
8	"function key 64"	prg	define function keys
9	"copier64 generat"	prg	generates next prog
2	"copy file 64"	prg	for single drive copies
71	blocks free.		

"transactor disk2"

1	"-vol 5 issue 01--"	prg	
2	"the brain"	prg	dazzler
2	"marquis 80"	prg	banner program
3	"marquis 64"	prg	
2	"marquis 40"	prg	
2	"marquis 20"	prg	
2	"the boxer"	prg	dazzler
3	"the plunge"	prg	dazzler
2	"sequins 40/80"	prg	dazzler
2	"sequins 64"	prg	
2	"curtains"	prg	dazzler
2	"graphic print"	prg	graphs weighted by letters
1	"reverse rvs"	prg	alternate rvs on/off
1	"1 line pet emulr"	prg	for most pet programs
3	"on error goto"	prg	error trapper
2	"but seriously"	prg	drum roll, cymbals
2	"zoundz"	prg	sid sound demo
1	"a-maze-ing"	prg	maze generator
5	"waves 64"	prg	sid surf sound
6	"vic 20 sound"	prg	bach for vic 20
22	"pet sound fx"	prg	popular sound effects
14	"sound maestro 64"	prg	sid sound utility
8	"quarter master"	prg	the song
9	"sprite rotate"	prg	
10	"quasimob loader"	prg	get more than 8 sprites
25	"quasimob.src"	prg	
2	"change screen 64"	prg	colour changes
7	"harmonic motion"	prg	plotting routine
6	"qtr sqr plot 80"	prg	quarter square plotter
6	"qtr sqr plot c64"	prg	
7	"qtr sqr vic norm"	prg	
7	"qtr sqr vic expd"	prg	
4	"projectile 4.0"	prg	plotting a projectile path
4	"projectile 64"	prg	
8	"talk clock 4.0"	prg	talking clock for use
9	"talk clock 64"	prg	with votrax sc-01
8	"talk clock vic"	prg	
3	"cartridge sim"	prg	simulate cartridges
5	"generate demo 64"	prg	program generator
2	"gen generate 64"	prg	
1	"-vol 5 issue 02--"	prg	
3	"down scroll 64"	prg	
1	"color ftoutsm 64"	prg	dazzler
5	"ftoutsm.ml"	prg	dazzler
1	"ama-zamara-ing"	prg	screen effects
1	"stop disable 64"	prg	disable run/stop restore
2	"autoliner 4.0 v2"	prg	generate line numbers
2	"autoliner 64 v2"	prg	version 2
2	"screen copy vic"	prg	screen to printer
2	"eep eep 4.0"	prg	basic 4.0 irq sound
2	"mirror 80"	prg	text mirroring
2	"mirror 40"	prg	
3	"mirror 64"	prg	
3	"ram scan 80"	prg	ML memory display
4	"ram scan 40"	prg	
3	"ram scan 64"	prg	
3	"crystal"	prg	draw crystalline patterns
2	"un cursor 80"	prg	
2	"un cursor 40"	prg	
3	"un cursor 64"	prg	
3	"un cursor 20"	prg	
1	"pop c64"	prg	clear returns from stack
4	"un token memory"	prg	a basic lister
5	"un token disk"	prg	
12	"merge 4.0"	prg	merge basic text
12	"merge c64"	prg	
39	"basic monitor 64"	prg	a basic MLM
11	"pi program"	prg	calculate pi
3	"rocket64 ml.run"	prg	ML thrust vs. gravity demo
7	"rocket64 bas.run"	prg	basic version

2	"rocket.sprt"	prg	
2	"rocket.obj"	prg	
8	"rocket listing 2"	prg	
10	"rocket listing 3"	prg	
8	"rocket listing 4"	prg	
3	"rocket listing 5"	prg	
6	"string insert 64"	prg	insert strings into strings
16	"cia timer demo"	prg	
7	"tod clock 64"	prg	time of day clock demo
5	"joycursor 64"	prg	control cursor with joystick
2	"sx64 emulate pt1"	prg	
1	"sx64 emulate pt2"	prg	
1	"-vol 5 issue 03--"	prg	
1	"line doo daa"	prg	dazzler
2	"colour test 64"	prg	contrast adjustment
1	"bytefinder ram"	prg	find byte values
1	"bytefinder disk"	prg	
3	"un-dim 4.0/2.0"	prg	clear arrays
4	"un-dim vic/64"	prg	
2	"errorrouter 64"	prg	trap errors
4	"line hider"	prg	hide entire lines
3	"ghost liner"	prg	hide lines but not line#
5	"list decorator"	prg	control over LIST
3	"etchasketch"	prg	
2	"default colours"	prg	after run/stop restore
2	"restore x 64"	prg	control DATA read pointer
2	"led demo"	prg	user port demo
3	"led knight rider"	prg	
4	"led roulette wheel"	prg	
15	"quadra 64 loader"	prg	partition text space
1	"quadra 64 init"	prg	
23	"quadra 64.src"	prg	
1	"basmon part2"	prg	ML monitor in basic
9	"picprint 64"	prg	hi-res to printer V1
17	"picprint 64.src"	prg	
4	"basic compare"	prg	compare basic prg files
8	"scramble.bas"	prg	basic encryption technique
9	"scramble.ml"	prg	
7	"scramble64.ml"	prg	
4	"password prot 1"	prg	password encryption
4	"password prot 2"	prg	file protection
10	"disk defender"	prg	disk protection demo
10	"disk defend 8050"	prg	
5	"lockdisk 4.0"	prg	force run after load
4	"lockdisk 64"	prg	
15	"drive protect"	prg	disk utility
7	"disk view/change"	prg	examine disk contents
16	"diskmod 4.0"	prg	
24	blocks free.		

"transactor disk3"

1	" + business & ed. + "	prg	
1	"quick beep 64"	prg	
1	"colour bar 64"	prg	compare colours
1	"dazzle attack4.0"	prg	dazzler
1	"aquarius 80"	prg	dazzler
3	"int scan 64.src"	prg	keyboard scanning
3	"int scan 4.0.src"	prg	
3	"file ripper 4.0"	prg	fast seq reader
3	"file ripper 64"	prg	
1	"file loader"	prg	successive loading demo
2	"ascii/cbm"	prg	converter
1	"magic number"	prg	arithmetricks
2	"safe val subtrn"	prg	VAL(substitute
3	"prime number gen"	prg	
1	"short prime gen"	prg	
4	"expeval 64 ldr"	prg	expression evaluator
4	"expeval plot"	prg	
7	"expeval 64.pal"	prg	

"transactor disk6"

1	"more aids + utils"	prg	
8	"aid4"	prg	
7	"vic aid.rel"	prg	
8	"c64 tiny aid ldr"	prg	
10	"supermon2.rel"	prg	
9	"supermon4.rel"	prg	
10	"super vicmon2"	prg	
10	"supermon64.v1"	prg	
10	"copy-all"	prg	
10	"copy-all64"	prg	
2	"copy file 64"	prg	
3	"transbasic instr"	prg	
4	"tb/add.obj"	prg	
15	"tb/kernel"	prg	
23	"tb/add.src"	prg	
10	"add"	prg	
5	"screen things"	prg	
4	"cursor position"	prg	TB: set or check crsr pos.
7	"set sprites"	prg	TB: sprite commands
5	"within"	prg	TB: test number for 'within range'
4	"read sprites"	prg	TB: sprite monitoring
4	"verifier.vic/64"	prg	
4	"verifier.pet"	prg	
10	"stp loader"	prg	seq to prg (batch) utility
1	"stp sys49152"	prg	
1	"stp sys828"	prg	
1	"....."	prg	
3	"racer pet"	prg	cheap video game
3	"racer 64"	prg	
3	"racer vic"	prg	
3	"racer + 4"	prg	
2	"tickertape 64"	prg	with sound effects
2	"dec to base b"	prg	convert to base b
2	"screen save vic"	prg	save screen to disk
3	"save@ exposed!!!"	prg	@ replace bug demo
13	"dos exec filer"	prg	execute programs in disk memory
2	"create device 9"	prg	device# changer and head knock
2	"create anti-nok"	prg	eliminator for use with above
34	"alpha dir.bas"	prg	alphabetize disk directories
26	"alpha dir.pal"	prg	
3	"auto-default.bas"	prg	change reset default values
4	"auto-def.create"	prg	
7	"auto-default.pal"	prg	
6	"file pursuit"	prg	trace and size
8	"supernumber.bas"	prg	indestructible variables
2	"supernum array"	prg	
11	"supernumber.pal"	prg	
1	"supnum sys49152"	prg	
1	"varptr"	prg	returns address of variables
4	"func keys.bas"	prg	screen editor extensions
4	"func keys.pal"	prg	
10	"bootmaker 64"	prg	makes boot for common loads
15	"datapoke aid"	prg	data entry utility
12	"load & run.pal"	prg	load and run ML programs
4	"l & r create.c64"	prg	automatically without
4	"l & r create.pet"	prg	necessarily knowing
4	"l & r create.vic"	prg	start addresses
10	"a/d pet.basic"	prg	analog to digital input
8	"a/d pet.pal"	prg	utility with hardware
6	"chopper"	prg	PAL file line splitter
13	"chopper.pal"	prg	
16	"labelgun"	prg	PAL label re-definition
31	"labelgun.pal"	prg	
44	"r65c02 assembler"	prg	for the R65C02 CPU
9	"fig 1 new ops"	seq	SEQ files for use with
1	"fig 3a writechrs"	seq	above program
4	"fig 3b writechrs"	seq	
2	"fig 4 linker eg."	seq	
135 blocks free.			

"transactor disk7"

1	"comm + networkin"	seq	
8	"aid4"	prg	
7	"vic aid.rel"	prg	
8	"c64 tiny aid ldr"	prg	
10	"supermon2.rel"	prg	
9	"supermon4.rel"	prg	
10	"super vicmon2"	prg	
10	"supermon64.v1"	prg	
10	"copy-all"	prg	
10	"copy-all64"	prg	
2	"copy file 64"	prg	
3	"transbasic instr"	prg	
4	"tb/add.obj"	prg	
15	"tb/kernel"	prg	
23	"tb/add.src"	prg	
10	"add"	prg	
5	"screen things"	prg	
4	"cursor position"	prg	
7	"set sprites"	prg	
5	"within"	prg	
4	"read sprites"	prg	
5	"strip & clean"	prg	TB: string purging
20	"scrolls"	prg	TB: scroll screen windows
4	"verifier.vic/64"	prg	
4	"verifier.pet"	prg	
10	"stp loader"	prg	
1	"stp sys49152"	prg	
1	"stp sys828"	prg	
1	"....."	prg	
4	"flash"	prg	flash characters for c64
2	"boxspiral + 4"	prg	plus4 box demo
4	"hires text"	prg	copy ROM chars to hi-res screen
1	"c64 fastkey"	prg	GET speed increaser
3	"banner"	prg	simple marquis program
1	"screen sizzle"	prg	dazzler
5	"b to xx32.bas"	prg	convert B machine files
3	"1541 align"	prg	1541 head alignment
7	"tdd termprog"	prg	TDD communications terminal
8	"remote.bas"	prg	c64 remote control utility
14	"remote.pal"	prg	
13	"c64 bbs link"	prg	bbs numbers database
3	"sort64"	prg	load this (,8,1) for use with above
16	"tele-tone 64"	prg	touch tone generator
5	"1541 dual drive"	prg	simulation using 2 1541s
11	"vector manager"	prg	cascade irq routines
2	"++++++"	prg	
8	"extra extra"	prg	instructions for following prgs
15	"terminal.r12"	prg	pet/cbm terminal prog
7	"term.r12"	prg	
4	"intelcom3"	prg	
4	"intelcom4"	prg	
1	"spet term"	prg	uses superpet acia
17	"term6s.g"	prg	
7	"terminal/vic.c1"	prg	vic 20 rs232 terminal
7	"term/vic.c1"	prg	
3	"firstdial3 boot"	prg	c64 with autodial
4	"firstdial3"	prg	
6	"mitedialer3 bt"	prg	c64 with mitey mo modem
3	"mitedialer3"	prg	
1	"firstterm3 boot"	prg	without autodial
54	"firstterm3"	prg	
3	"firstdial3 print"	prg	
132	"firstdial3 doc"	seq	documentation
3	"firstdial3 fkeys"	seq	fkey definitions
1	"higgyboot + 4"	prg	plus 4 terminal
41	"higgyterm + 4"	prg	
28 blocks free.			

"transactor disk8"			"transactor disk9"		
1	"the languages"	seq	1	"impl. sciences"	seq
8	"aid4"	prg	8	"aid4"	prg
7	"vic aid.rel"	prg	7	"vic aid.rel"	prg
8	"c64 tiny aid ldr"	prg	8	"c64 tiny aid ldr"	prg
10	"supermon2.rel"	prg	10	"supermon2.rel"	prg
9	"supermon4.rel"	prg	9	"supermon4.rel"	prg
10	"super vicmon2"	prg	10	"super vicmon2"	prg
10	"supermon64.v1"	prg	10	"supermon64.v1"	prg
10	"copy-all"	prg	10	"copy-all"	prg
10	"copy-all64"	prg	10	"copy-all64"	prg
2	"copy file 64"	prg	2	"copy file 64"	prg
3	"transbasic instr"	prg	17	"yellow pages 1.2"	prg c64 disk utility
4	"tb/add.obj"	prg	3	"transbasic instr"	prg
15	"tb/kernel"	prg	4	"tb/add.obj"	prg
23	"tb/add.src"	prg	15	"tb/kernel"	prg
10	"add"	prg	23	"tb/add.src"	prg
5	"screen things"	prg	10	"add"	prg
4	"cursor position"	prg	32	"use"	prg TB: like add, only faster
7	"set sprites"	prg	5	"screen things"	prg
5	"within"	prg	4	"cursor position"	prg
4	"read sprites"	prg	7	"set sprites"	prg
5	"strip & clean"	prg	5	"within"	prg
20	"scrolls"	prg	4	"read sprites"	prg
10	"labels"	prg TB: basic code labelling	5	"strip & clean"	prg
5	"token & var"	prg TB: token\$(, var(	20	"scrolls"	prg
8	"instring"	prg TB: string compare	10	"labels"	prg
7	"place"	prg TB: scan strings	5	"token & var"	prg
7	"arcfunctions"	prg TB: arcsine, arccos	8	"instring"	prg
3	"printat"	prg TB: print@ col,row	7	"place"	prg
18	"sound things"	prg TB: sid control	7	"arcfunctions"	prg
4	"verifizer.vic/64"	prg	3	"printat"	prg
4	"verifizer.pet"	prg	18	"sound things"	prg
10	"stp loader"	prg	12	"move & fill"	prg TB: memory transfers
1	"stp sys49152"	prg	11	"dos support"	prg TB: dos command wedge
1	"stp sys828"	prg	5	"line calc"	prg TB: for gotos & gosubs
2	"RUN ME!"	prg prints subscription form	3	"beep"	prg TB: SID beep
3	"RUN ME TEXT"	seq	3	"stripper"	prg removes PAL comments
12	"RUN ME FORM"	seq	4	"verifizer.vic/64"	prg
1	"-----"	prg	4	"verifizer.pet"	prg
1	"1541 spin1"	prg spin 1541 drive motor	10	"stp loader"	prg
2	"1541 spin2"	prg uses SHIFT for on/off	1	"stp sys49152"	prg
1	"blks free-1541"	prg reports 1541 blocks free	1	"stp sys828"	prg
2	"save-protect"	prg 1541 BAM adjuster	2	"RUN ME!"	prg
1	"scratch&save.bas"	prg SAVE@ alternative	3	"RUN ME TEXT"	seq
4	"scratch&save.pal"	prg	12	"RUN ME FORM"	seq
4	"menu prg"	prg menu driver	1	"-----"	prg
2	"list freeze"	prg	6	"quake"	prg dazzler
2	"waving spokes"	prg plus4 graphics demo	5	"errcat 64/20"	prg catalog routine
2	"kaleidoscope"	prg dazzler	2	"rt justify pet"	prg right justify demo
3	"asc to bin.pal"	prg in ML	2	"drive speed"	prg alter drive parameters
2	"bin to asc.pal"	prg	2	"basic stp"	prg seq to prg in basic
1	"lett'er fly!"	prg GET tips	3	"gauss elim"	prg gaussian elimination routine
3	"18-0 un-screw"	prg disk utility	4	"lottery"	prg lottery numbers generator
4	"star 0.14"	seq for COMAL 0.14	1	"swords of doom"	prg dazzler
4	"star 2.00"	seq for cartridge COMAL 2.00	1	"sum of squares 1"	prg summing techniques to
4	"profiler.pal"	prg analyze time consumption	1	"sum of squares 2"	prg help overcome problems
2	"profiler.bas"	prg of basic programs	1	"sum of squares 3"	prg with binary inaccuracies
3	"profiler.ldr"	prg	11	"projector"	prg plot graphs with 3 dimensions
4	"hi-res text demo"	prg print scaled hi-res text	9	"hires"	prg for use with projector
12	"hi-res text ldr"	prg	13	"bigprint"	prg for use with projector
23	"hi-res text src"	prg	6	"timer64.bas"	prg microsecond timer
2	"+++++++"	prg	4	"timer64.pal"	prg
5	"bootcomal"	prg boot for COMAL 0.14	3	"projectile pet"	prg bounce plotter
131	"comal80.can"	prg notice the block count	4	"projectile 64"	prg bouncing sprite plotter
5	"comalerrors"	seq	12	"comp1.pal"	prg data compressor programs
10	"genererror.e.l"	seq	20	"comp2.pal"	prg
6	"hi"	prg	2	"comp1.obj",8,8:	prg
6	"see'information"	prg	2	"comp2.obj",8,8:	prg
6	"see'instructions"	prg	5	"koala split"	prg for use with compressor, splits
21	"information83nov"	seq	6	"anim split"	prg koala/animation station pictures
42	"instructions0.14"	seq	2	"vars-indestruct"	prg edit without losing variables
20	"logo'book'sample"	prg	41	"unassembler"	prg make source code from object
37 blocks free.			10	"super sound"	prg SID sound utility
			116 blocks free.		

News BRK

Submitting NEWS BRK Press Releases

If you have a press release which you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way Applicable to Commodore equipment.

Transactor News

Transactor Subscription Prices...

Are remaining at the old price of only \$15.00 per year, even with the new Newsstand price of \$3.50/copy. Now a subscription to The Transactor saves you even more - 29% over the newsstand price! Now is the time to finally get around to sending in that subscription card you've been holding on to.

Viewtron Starter Kit

The Viewtron Starter Kit will be available in Canada from Transactor Publishing. The price is about the same after the exchange rate, but experience has shown that ordering goods from the U.S. into Canada can take longer than migrating tortoises. Also, phone orders from Canada mean a long-distance call to Miami since toll-free 1-800 numbers don't cross international borders. And speaking of borders, well... we all know what happens to packages that even look dutyable.

The Starter Kit is a nifty package, too. The software is terrific. It comes with files containing several phone numbers for the major networks and a program that displays those within your area code. Once your ID

and Password are recorded, signing on is simply a matter of loading the Viewtron communications program and hitting one key. You'll like the manual - 46 pages of handy reference and instructions for everything from using Viewtron CB to dumping to printer, disk, etc. For more on Viewtron, see the article in this issue.

So if you live in the U.S., call Viewtron's toll free line (see next item). But if you're ordering the kit for delivery within Canada, you can use our postage paid order form and let us arrange for the first half of it's journey.

Viewtron Now Available To Commodore Owners

Viewdata Corporation of America is now offering its Viewtron videotex service to Commodore personal computer owners in most cities throughout Canada and the U.S.

Viewtron subscribers can save time gathering valuable information. They can get up-to-the-minute news, weather forecasts, continuous sports scores, current stock prices, airline schedules and fares, consumer reports and movie and book reviews.

Viewtron subscribers can also send electronic messages, and, through a growing list of banks, pay bills and get account balances. They even have access to an up-to-date encyclopedia.

In addition to the basic Viewtron service, there are enhanced services for Commodore subscribers. Some of these include:

- Commodore software reviews and ratings
- Software and hardware advice from experts and other subscribers
- Discounts on top-selling Commodore software
- Discounts on computer hardware
- Commodore special interest clubs

Viewtron is the first complete on-line service that runs in colour on Commodore 64s. To get Viewtron, Commodore owners will need to purchase the Viewtron Software Starter Kit for \$9.95 (US). The kit includes a diskette with communications software, one free hour of service, an ID and password, and a user manual.

Subscribers will pay only for their use of the service. No monthly minimums will be charged. Rates are: nine cents a minute, weekday nights (after 6 p.m.) and weekends; and 22 cents a minute weekdays. And, unlike other services, there is no extra charge for subscribers using 1200-baud modems. (Prices may vary in some cities.)

Viewtron also runs on the Commodore 128, and will run on the Amiga in 1986. For more information, call the Viewdata Corporation at 1-800-272-5400

More Viewtron News

Commodity traders almost anywhere in the U.S. can now keep up with the volatile and complex futures market with VIEWTRON®, the electronic information service offered by Viewdata Corporation of America, Inc.

VCA offers trading prices of nearly 100 commodities, ranging from aluminum to wheat, updated every 10 minutes. Prices are gathered from 14 exchanges by Commodity News Services, Leawood, Kansas. Both companies are owned by Knight-Ridder Newspapers.

When a subscriber requests a price quotation, all of the contracts in a given commodity are listed, with their opening, high, low and last available prices, along with the change from the previous settlement price.

Contact: John L. McCarthy,
V.P. Business Information Services
(305) 674-3499

West Coast Commodore Show II

The West Coast Commodore Association proudly announces "THE COMMODORE SHOW II" to be held on Saturday, February 8th and Sunday, February 9th, 1986 from 10:00 am to 6:00 pm at the Cathedral Hill Hotel in San Francisco.

This Commodore-specific trade show will feature the latest in software and hardware for the vast Commodore users market. THE COMMODORE SHOW II will cover the newest Commodore machines; the Amiga and the C-128 as well as the 64 and Plus 4.

The show will also feature noted Commodore experts speaking on graphics, telecommunications, music, business applications and other subjects of interest to Commodore users.

The first COMMODORE SHOW held in February of 1985 drew 5200 attendees and next year's show promises to be bigger and better.

For information on booth space and advance ticket sales, interested users groups and hardware/software vendors should contact:

West Coast Commodore Association
P.O. Box 210638
San Francisco, CA 94121
(415) 982-1040

Commodore 128 On Dealer Shelves

Toronto, Ont. — Commodore Business Machine Limited announced that the Commodore 128 Personal Computer is in stores across Canada effective October 1. Over 1,000 retail outlets including mass merchandisers, department stores and computer dealers, will carry the new C128.

The Commodore Ham's Companion

Springfield, Illinois — QSKY Publishing introduces a new book designed to fill the need for information on using Commodore computers in the amateur radio "ham shack". "The Commodore Ham's Companion", written by Jim Grubbs, K9EI, helps show the way toward effective use of Commodore machines for this purpose.

The book's fourteen chapters address many subjects, including:

- Selecting a Commodore machine for the ham shack, or upgrading your present system.

- The basics behind programming for data including RTTY, Morse, AMTOR and Packet.
- The ins and outs of information management, like log, dupe and awards programs, are explained.
- Why Commodore machines are the easiest to interface
- "Telehamming" - connecting to amateur radio information by telephone line.
- Where to find specialized software for slow scan television, satellite tracking and many other exotic applications.
- How to obtain a dramatic increase in speed without learning machine language.

Additionally, a set of valuable appendices include over 80 sources for software and hardware for amateur radio applications. A bibliography with over 60 magazine articles and columns on Commodore computers in the radio shack is also included. A glossary and other resource list rounds out the book.

Retail price for the 160-page paperback is \$15.95 U.S. plus \$2.50 for shipping and handling. For more information or to order, contact:

QSKY Publishing
P.O. Box 3042
Springfield, IL
USA 62708

Starting Your Computer Services Business

J.V. Technologies, Inc. announces the release of the new book *Starting Your Computer Services Business* by Dr. John Desiderio for those interested in using their computer to earn extra income. This comprehensive manual discusses the various phases and stages of starting a business, examining pitfalls and reviewing the proper steps to successfully organizing a new business.

Various business formats, such as, sole proprietorships, partnerships and corporations, are examined in detail with a clear discussion of the advantages and disadvantages of each. The book not only reviews the typical computer services of consulting and word processing, but extends beyond these to discuss how activities such as teaching, telecommunications, article and book writing, software development, etc. can also contribute to starting and maintaining a flourishing computer services business. Persons interested in working

either part- or full-time, from their home or in a small office, will find this book to be an invaluable resource.

Also included is a separate chapter on advertising and a complete section of reproducible forms for keeping track of your business activities, such as advertisements, orders, etc. The introductory cost of the publication is \$9.95 (US) and a 30 day money-back guarantee is also offered. Contact:

J.V. Technologies, Inc.
P.O. Box 563
Ludington, MI 49431
(616) 843-9512

1986 Printer Directory and Specification Guide

Gorham International is initiating the "1986 Printing Directory and Specification Guidebook Series". The series consists of four volumes:

TITLE	Scheduled Publication Date
Thermal Printing	Oct. 31, 1985
Toner Based Printing	Nov. 30, 1985
Colour Hard Copy Printer	Dec. 31, 1985
Ink Jet Printing	Jan. 31, 1986

Primary features of this four volume series include:

- Specifications of each printer system presented in a standard, easy to use format from Gorham's computerized database
- Inclusion of actual manufacturer's product literature and print samples in all possible cases.
- Gorham's computerized database provides a detailed summary and overview section in each publication. This enables useful categorizations of competitive products and grouping of products by selected specifications, applications, etc.
- Hardbound reference volumes designed to aid current and potential users in their identification, evaluation and selection of printer products. These limited edition publications of 500 copies each provide a unique presentation format by which product manufacturers can provide practical information, literature and samples to marketplace.

Vicki Woodbrey, who manages Gorham's database, states that "all manufacturers and suppliers of hardware, consumables, components, services, etc. are encouraged to contact her to assure their product information has been incorporated into the Gorham database."

Each of the four volumes are available individually at a price of \$250.00 U.S. including shipping for prepaid orders (additional \$25.00 for overseas airmail). For additional information or to verify product inclusion, please contact:

Alvin G. Keene or Vicki Woodbrey
GORHAM INTERNATIONAL INC.
P.O. Box 8
Gorham, ME 04038

Scenery Disks Now Available for Flight Simulator II and Jet

SubLOGIC is pleased to announce the release of six different Scenery Disks for the Commodore 64 and IBM PC. These Scenery Disks expand the potential flying environment of SubLOGIC flight simulation products including Flight Simulator II, Jet, and the (IBM) Microsoft Flight Simulator.

Six Scenery Disks covering the entire western half of the Continental United States are now available. Each disk covers a geographical region of the country, and includes the major airports, radio-nav aids, cities, highways, rivers, and lakes located in that region. Enough detail is included on each disk for either visual or instrument cross-country navigation.

Each Scenery Disk package comes complete with appropriate sectional charts plus full airport and nav-aid directories. Individual Scenery Disk packages are available for \$19.95 (US) each. The Western U.S. six-disk set, packaged in a vinyl three-ring notebook with dividers, may be purchased for \$99.95 (US).

To order, specify computer system and which disks you want, add \$2.00 (\$5.00 for the six-disk set) for postage, and specify UPS or first class mail delivery.

SubLOGIC Corporation
713 Edgebrook Drive
Champaign, IL 61820
(217) 359-8482
Order Line: 1-800-637-4983

ZIPP-CODE-48 Development System For C-64

Now you can write code on the Commodore 64/128 for smart peripherals, robotics and other applications using the 8048/8748 family of microcontrollers.

The ZIPP-CODE-48 Cross Assembler is a powerful, symbolic assembler for the 8048 microprocessor family. In addition to the basic instruction set used with the 8048/8748 NMOS microprocessors, the ZIPP-CODE-48 System correctly cross-assembles, disassembles, and simulates four other related instruction sets. (8021, 8022, 80C48/49/50.)

Additional features of the assembler are: a built-in simulator for debugging; works with standard BASIC-format source files; excellent error-detection; works with tape or disk, with or without printer.

Extra utilities included with the package are a print utility with with alphabetical cross-reference symbol table; a line disassembler; a machine-code save utility; and a full-featured machine code monitor program.

ZIPP-CODE-48 is available for \$49.95 (US). Order from:

Hughes Associates Software
45341 Harmony Lane
Belleville, MI 48111
(313) 699-1931

Automated Telecommunications Package For The 64 and 128

Progressive Peripherals & Software, Inc. of Denver, Co. has announced the introduction of BOBSTERM PRO, a uniquely powerful telecommunication software program for the Commodore 64 and 128 personal computers. BOBSTERM PRO is unique because it allows complete user control over every aspect of telecommunications.

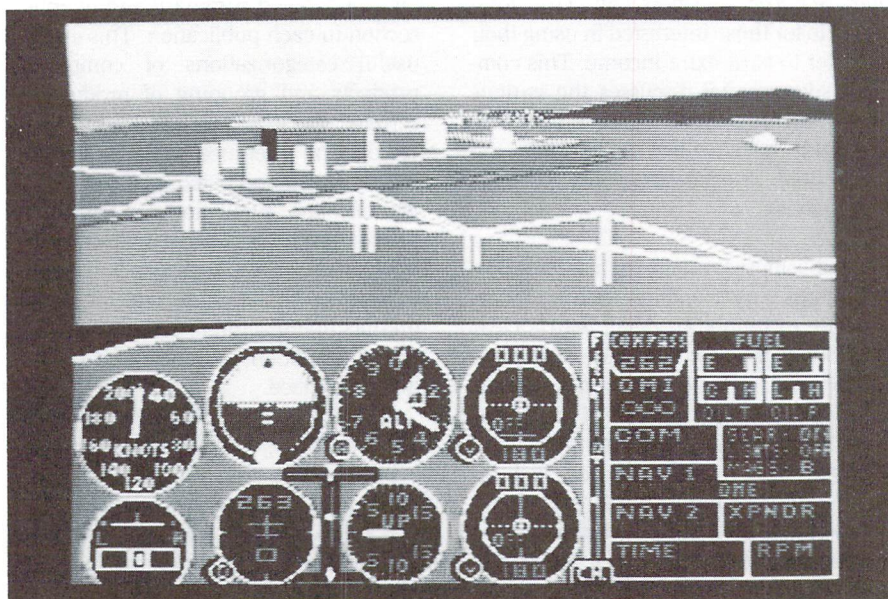
Every feature of BOBSTERM PRO can be automated through the use of macro programming. Log-on, password entries, auto-start, auto-answer, auto-dial, and transmission of files are a few of the many operations that can be reduced to a single key stroke. Linked together, these macro commands make continuous BBS maintenance simpler and a great deal faster. Complete screen editing, formatting, screen colours and status line information are available to the user. Seven custom character sets are built in for hard copies, with space reserved for three user-defined fonts.

BOBSTERM PRO is compatible with nine popular modems and can be easily adapted to many more. Data transfer can be accomplished through the use of straight ASCII, straight binary, sequential line with prompt wait, XON/XOFF, Punter protocol, XMODEM protocol, or entire disk (C64 to C64). With the special FILL feature, the normal 28.5 K memory can be expanded to handle 94 K files. BOBSTERM PRO is also compatible with all Commodore and second party computer hardware. Contact:

Progressive Peripherals & Software, Inc.
464 Kalamath Street
Denver, CO 80204
(303) 825-4144

More 64 Software From Progressive

Progressive Peripherals & Software, North American agent for Precision Software of Surrey, England, is pleased to announce SUPERBASE STARTER, a user friendly software database for the Commodore 64 and 128 personal computers.



SUPERBASE STARTER is an electronic filing cabinet to store, retrieve, update, display, and print information in a multitude of ways. SUPERBASE STARTER is a beginning version of the bestselling SUPERBASE. The program runs with application starter packs, making it ideal for non-technical users. The package comes with a manual written with this user in mind. Easy to understand menus of commands and built-in Help Screens are designed to ease the user into the program with a minimum of stress. An audio-learning cassette is also available, upon registration, as an excellent aid to learning SUPERBASE STARTER.

Another addition to the growing line of "SUPER" software programs is SUPERTYPE for the CV-64 and 128. SUPERTYPE is a complete educational tool, addressing the developmental needs of the information age. SUPERTYPE builds touch typing skills in 19 tiered lessons, each leading to the next. Colour tabs are provided to delineate each finger's relationship to a group of keys on the keyboard. A built-in metronome encourages proper rhythm to increase speeds.

SUPERTYPE provides accurate assessment of the student's progress through 'Results Screens' at the end of each session. Times are calculated in sentences typed, errors during exercises, and words per minute corrected and uncorrected. SUPERTYPE then points out which fingers need to be practised and which fingers are making the most mistakes. The steno-style SUPERTYPE manual stands alone, so the student can progress easily through the lessons without hunting for instructions.

Progressive Distributes Commodore's 8023P Printers

Progressive Peripherals & Software, Inc. has become a major distributor for Commodore's 8023P Printer. After successfully opening the market for Commodore's SFD 1001 disk drive, Progressive has decided to distribute the 8023P Printer. In the Commodore microcomputer industry, the 8023P Printer is best known for its high speed and 15 1/2 inch wide carriage. The printer has both business applications and home use advantages. The 8023P Printer is ideal for business use, as it features these options: condensed print, Commodore graphics, wide carriage for printing out large spread sheets, and 150 CPS high speed. The advantages for home use of the 8023P Printer are a near letter-quality

mode, longevity, and it is twice as sturdy as the average printer. The 8023P Printer is fully compatible with word processing, database, and other business and personal software. The printer will retail for \$299.00 (US).

Progressive Releases E-Link

Progressive Peripherals & Software, Inc. is shipping their newly released Commodore serial to IEEE interface. The E-Link is designed for compatibility with Commodore IEEE peripherals and is totally transparent to the Commodore 64. According to Kris Halverson, Product Manager, "There is a strong demand for an interface that gives the consumer access to the many high-powered Commodore peripherals on the market. The customer will need only one E-Link regardless of the number of IEEE peripherals in use."

Halverson used computer aided design to ensure the rugged reliability of E-Link. E-Link has the following features: uses no internal C-64 memory; independent power supply; and microprocessor controlled. The response to E-Link by Commodore dealers has been strong because E-Link provides needed accessibility to the many available Commodore disk drives and printers. The interface will retail for \$99.95 (US). Contact:

Progressive Peripherals & Software, Inc.
2186 South Holly
Denver, Colorado 80222
(303) 759-5713

Low Cost Temperature Monitoring For The Commodore 64

Applied Technologies, Inc. announces a breakthrough in low cost data logging and temperature monitoring with Commodore computers.



Features include display of 8 or 16 temperature channels, temperature range of -15 degrees to +180 degrees Fahrenheit at approximately 1 degree resolution, electronic interface plugs directly into the joystick port, inexpensive weather-protected sensors, and menu-driven software included.

The 8-channel system starts at \$89.95 (US). For more information, contact:

Applied Technologies, Inc.
Computer Products Division
Lyndon Way, Kittery, Maine 03904

Communications Chips Seen Rising in Sales As Semiconductor Industry Continues to Slow

Norwalk, CT — Very few people still believe that 1985 will be a good year for the semiconductor industry. In fact, some estimates say the market will be down close to 20 percent this year alone. But integrated circuits for communications equipment will see a 30 percent average growth rate for the rest of this decade, according to a just-published 157-page research report from International Resource Development Inc., a Norwalk, CT market research and consulting firm.

The report, entitled *Telecommunications Integrated Circuits* (#659), covers over fifty major players in the communications IC marketplace, and predicts that even when growth resumes in the semiconductor market as a whole, telecommunications and data communications integrated circuits will continue to outpace the general chip market.

Further details of the \$1,650.00 report, including a free table of contents and a description of the topics covered, are available from:

International Resource Development Inc.
6 Prowitt Street
Norwalk, CT 06855
(203) 866-7800; Telex 64 3452

**PAYS
\$40**

per page for articles

We're also looking for
professionally
drawn cartoons!

Send all material to:

The Editor
The Transactor
500 Steeles Avenue
Milton, Ontario
L9T 3P7

Volume 6 Editorial Schedule

Issue#	Theme	Copy Due	Printed	Release Date
1	More Aids & Utilities	Feb 1	Mar 22	April 1/85
2	Communications & Networking	Apr 1	May 24	June 1
3	Languages	Jun 1	Jul 26	August 1
4	Implementing The Sciences	Aug 1	Sep 20	October 1
5	Hardware & Software Interfacing	Oct 1	Nov 22	December 1
6	Real Life Applications	Dec 1	Jan 24	February 1/86

Volume 7 Editorial Schedule

1	ROM Routines / Kernel Routines	Feb 1	Mar 21	April 1
2	Games From The Inside Out	Apr 1	May 23	June 1
3	Programming The Chips	Jun 1	Jul 25	August 1
4	Gadgets and Gizmos	Aug 1	Sep 26	October 1
5	Simulations and Modelling	Oct 1	Nov 21	December 1
6	Programming Techniques	Dec 1	Jan 23	February 1/87

Advertisers and Authors should have material submitted no
later than the 'Copy Due' date to be included
with the respective issue.

COMAL INFO If you have COMAL— We have INFORMATION.

BOOKS:

- COMAL From A To Z, \$6.95
- COMAL Workbook, \$6.95
- Commodore 64 Graphics With COMAL, \$14.95
- COMAL Handbook, \$18.95
- Beginning COMAL, \$22.95
- Structured Programming With COMAL, \$26.95
- Foundations With COMAL, \$19.95
- Cartridge Graphics and Sound, \$9.95
- Captain COMAL Gets Organized, \$19.95
- Graphics Primer, \$19.95
- COMAL 2.0 Packages, \$19.95
- Library of Functions and Procedures, \$19.95

OTHER:

- COMAL TODAY subscription, 6 issues, \$14.95
- COMAL 0.14, Cheatsheet Keyboard Overlay, \$3.95
- COMAL Starter Kit (3 disks, 1 book), \$29.95
- 19 Different COMAL Disks only \$94.05
- Deluxe COMAL Cartridge Package, \$128.95
(includes 2 books, 2 disks, and cartridge)

ORDER NOW:

Call TOLL-FREE: 1-800-356-5324 ext 1307 VISA or MasterCard
ORDERS ONLY. Questions and Information must call our
Info Line: 608-222-4432. All orders prepaid only—no C.O.D.
Add \$2 per book shipping. Send a SASE for FREE Info
Package or send check or money order in US Dollars to:

COMAL USERS GROUP, U.S.A., LIMITED
5501 Groveland Ter., Madison, WI 53716

TRADEMARKS: Commodore 64 of Commodore Electronics Ltd.;
Captain COMAL of COMAL Users Group, U.S.A., Ltd.

JOIN TPUG

The largest Commodore Users Group

Benefit from:

Access to library of public domain software
for C-64, VIC 20 and PET/CBM

Magazine (10 per year) with advice from

Jim Butterfield
Brad Bjomdahl
Liz Deal

TPUG yearly memberships:

Regular member (attends meetings)	—\$35.00 Cdn.
Student member (full-time, attends meetings)	—\$25.00 Cdn.
Associate (Canada)	—\$25.00 Cdn.
Associate (U.S.A.)	—\$25.00 U.S.
	—\$30.00 Cdn.
Associate (Overseas — sea mail)	—\$35.00 U.S.
Associate (Overseas — air mail)	—\$45.00 U.S.

FOR FURTHER INFORMATION:

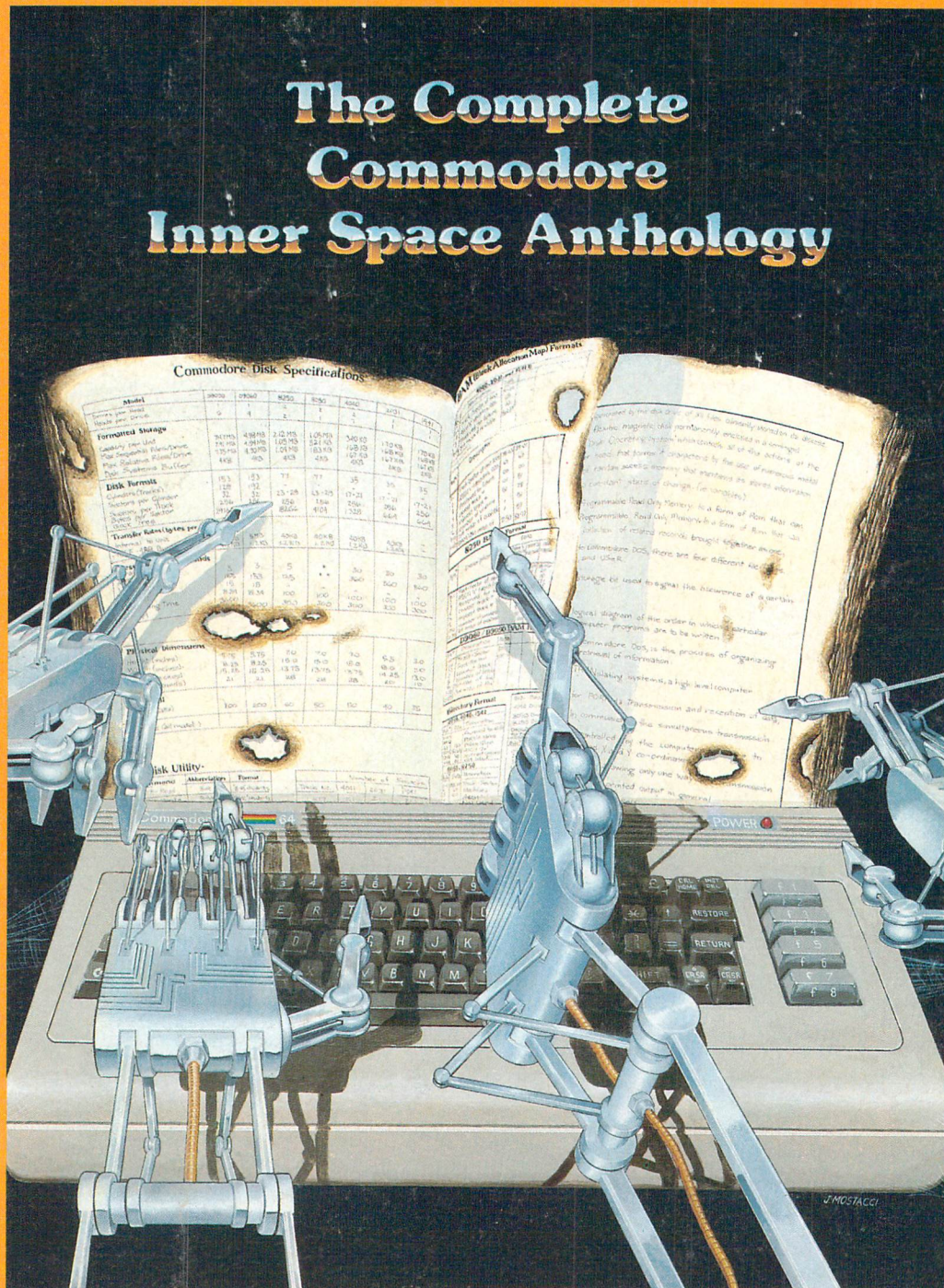
Send \$1.00 for an information catalogue
(tell us which machine you use!)

To: TPUG INC.

DEPT. A,
101 DUNCAN MILL RD., SUITE G7
DON MILLS, ONTARIO
CANADA M3B 1Z3



The Transactor presents, The Complete Commodore Inner Space Anthology



Over 7,000 Delivered Since March '85

Postage Paid Order Form at Center Page